Technical University Munich
Faculty for civil engineering and land surveying
Remote Sensing Technology
Prof. Dr.-Ing. Richard Bamler

Solving Optimization and Inverse Problems in Remote Sensing by using Evolutionary Algorithms

Dipl.Ing. (FH) Peter Fischer

Master Thesis

Editing: 5.11.2012 – 3.05.2013

Study Course: Geodesy and Geoinformation (Master)

Advisor: Diego Loyola, Dr.-Ing. Stefan Auer

2013

Contents

List of Figures

List of Tables

Abstract

This thesis objective is the solving of combinatorical and inverse problems in Remote Sensing by using genetic algorithms. The first part introduces optimization theory. Four different deterministic local search algorithms are reviewed. Differences and similarities between these algorithms are examined, also their behavior in a representative test domain. Then the theory of evolutionary computing is explained. It is shown, that evolutionary algorithms are in contrast to the previous discussed search algorithms not deterministic, but heuristic. Furthermore the difference between local and global search is pointed out. An genetic algorithm, which is inspired by evolutionary algorithms, is developed. The program is written in an object oriented style using C++. This program is tested with several test functions which are common in global optimization literature. But besides of forward problems, also an inverse problem is solved in this methodology part. It is shown that the algorithm delivers reasonable results. The algorithm is enhanced with local search and parallel computation. The proof is made that by merging local and global search, a significant reduction in the number of function calls can be reached. Moreover by doing hybridization more robust results are gained. At the end of the methodology part the reader has an overview about the developed genetic algorithm and the different search strategies.

In the second part two problems in the field of Remote Sensing are solved using genetic algorithms. The first one is a combinatorical task, which arises in the field of an ozone retrieval algorithm. The parallelized genetic algorithm is adapted to the specific problem domain. The fitness function is formulated according to the combinatorical problem, methods are written for the specific tasks like reading HDF files and starting external processes. Then under different conditions the program is applied and the results are discussed.

A second problem deals with the retrieval of cloud parameters. This task is an inverse problem and the genetic algorithm is enhanced with an local search operator. The task is about finding input parameters that correspond to given measurements. Because of this, a total least squares approach is selected for the local search. As a result we see that the hybrid approach provides more accurate results then the pure genetic algorithm.

1 Introduction

Optimization is one of the very first topics pupils get in touch with in their high school math lessons. Even if the term optimization is not used, the driving idea for solving a mathematical problem is an optimization problem. The first problems are about finding the zero point in curve discussions, or answering questions like

> A baker has x gram meal and y gram barm available. How much buns and prezels can he produce maximal, if a bun needs xx gram meal and yy gram barm and a pretzel needs xx gram meal and yy gram barm?

Later in university engineering courses the gained knowledge is applied to various specific problems, for example fitting of data to a model by minimizing the residuals between the data and the model. But in contrast to the simple school problems, the real world models which engineers solve are now more complicated. Terms like linear and non-linear problems, constraints, multi-dimensionality and much more arise. To succeed in these problem domains students have to know a bigger set of optimization algorithms, which are often quite different in their behaviors. The clue is to realize that there's no "One fit's all" solution in optimization theory. It's up to the engineer to find the right algorithm for the actual problem. A deeper understanding in optimization theory and the problem domain is mandatory.

This thesis focuses on the analysis of optimization algorithms - deterministic, heuristic and the combination of both, hybrid algorithms. The field of application is Remote Sensing. Prior to real-world application each of the algorithms is applied to test functions.

The idea of combining deterministic and heuristic algorithms is simply explained by the results they produce. The classical deterministic algorithms produce local solutions, mostly with fast convergence. The heuristic approach, in this thesis a genetic algorithm with evolution strategy, tends to find the global solution with slower convergence. By combining both, we want to guarante that our results always represent the global minimum, furthermore the used computational effort in terms of function calls should be minimal.

In the first part of this thesis the deterministic algorithms are introduced and their behavior is explained by applying them to low-dimensional problems. Tables show how these algorithms converge to their results numerically, figures visualize the iterative process.

The second part introduces the genetic algorithm. Besides of the evolution theory which is the driving power of this algorithms, the application of this algorithm to an broad range of problems

is shown.

The third part gives an idea of how both algorithm families, deterministic and heuristic, can take profit from each other. This is done by fusing the previous algorithms by means of programming. The new hybrid algorithm is then applied to the problems of the section before, to show whether this approach is really beneficial.

At the end of the introduction the programming aspect is briefly discussed. The genetic algorithm is an own development in C++. For the deterministic algorithms, and also for random numbers, sorting and other functions, the GNU Scientific Library is used. Furthermore the Boost Library is included for solving system functions like multi threading and the execution of external processes.

For non programming tasks like analyzing log files and plotting the GNU Octave software is used.

The last part is about application. Here we show a combinatorial and an inversion problem, which are both related to atmospheric remote sensing. The combinatorical task is in theory quite different to the problems solved in the test environments, here the strength of genetic algorithms can be fully used. The inversion problem uses a radiative transfer model, which also includes the computation of the partial derivatives for the unknown. We apply here an hybrid approach.

2 Optimization methods

At the beginning some basic naming conventions and definitions have to be introduced. Especially the understanding of the term "Optimization" has to be clarified. Nocedal and Wright gave an intuitive explanation by using the following terms [3]:

- *objective* - the *objective* could be the needed time to do a movement, the price of a product or any combination of quantities that can be represented by a single number.
- *variables/unknowns* - the *objective* depends on the *variables/unknowns*, the goal is to find the *variables/unknowns* that optimize the *objective*.
- *constrains* - often the *variables* are restricted, or *constrained*, in some way. For example, the time needed for producing a product can't be negative.
- *modeling* - the process of identifying the objective, the variables and the constrains is called *modeling*. It's the first and most important step in the optimization process, because the solution of a problem can only as good as the describing model.
- *optimality condition* - when the model is set, an algorithm solves the optimization task. After the application of the algorithm to the model, we need to identify whether the algorithm has succeeded in finding a solution. Often there are mathematical expressions called *optimality conditions* for checking whether the current set of variables is the solution of the problem.

Besides of this general definition a more strict, mathematical formulation can be done.

> Optimization is the minimization or maximization of a function subject to constraints on its variables.

The following example, adapted from Nocedal [3], fits well for the purpose of illustrating an optimization task. The following notation is used:

- x is the vector of *variables*, also called *unknowns* or *parameters*
- f is the *objective function*, a (scalar) function of x that we want to maximize or minimize
- c_i are *constraint* functions, which are scalar functions of x that define certain equations and inequalities that the unknown vector x must satisfy

Using this notation, the optimization problem can be written as follows:

$$\min_{x \in \mathbb{R}^n} f(x) \text{ subject to } \begin{cases} c_i(x) = 0, i \in \mathcal{E} \\ c_i(x) \geq 0, i \in \mathcal{I} \end{cases} \tag{2.1}$$

$\mathcal{E}$ and $\mathcal{I}$ are sets of indices for equality and inequality constraints. A simple example follows, also adopted from Nocedal and Wright, which helps to visualize these terms. Let's consider the problem[1]

$$\min (x_1 - 2)^2 + (x_2 - 1)^2 \text{ subject to } \begin{cases} x_1^2 - x_2 \leq 0 \\ x_1 + x_2 \leq 2 \end{cases} \tag{2.2}$$

We can write this problem in the form

$$f(x) = (x_1 - 2)^2 + (x_2 - 1)^2,\ x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},\ c(x) = \begin{bmatrix} c_1(x) \\ c_2(x) \end{bmatrix} = \begin{bmatrix} -x_1^2 + x_2 \\ -x_1 - x_2 + 2 \end{bmatrix},\ \mathcal{I} = \{1, 2\},\ \mathcal{E} = \varnothing \tag{2.3}$$

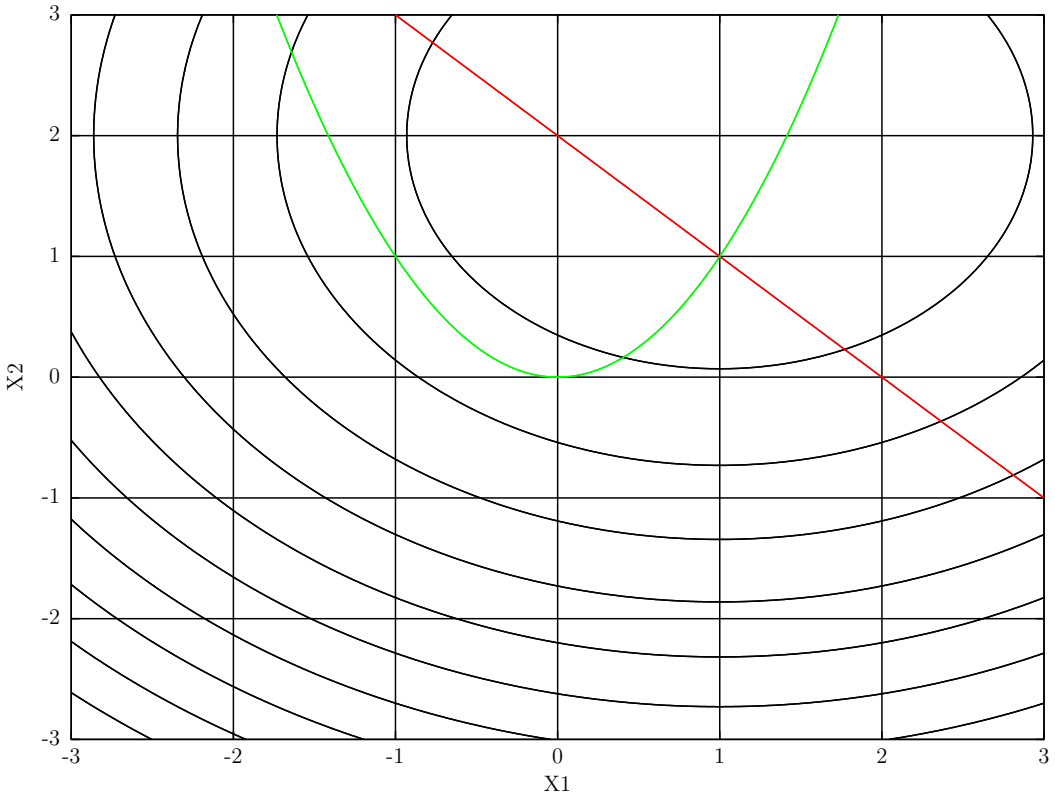


Figure 2.1 – Contour plot of function 2.1 with constraints 2.2

[1] example function 2.1 taken from Nocedal and Wright, page 2

Figure 2.1 gives an overview of the problem domain. The black lines represent the contours of $f(x)$, where the function has constant values. The green line represents the border contour of c_1 where the inner part fulfills the constraint. The red line represents the border contour of c_2 where the left side fulfills the constraint.

The following problems are all unconstrained. The important thing which should be clear now are the namings and their related meanings. Later on the *variables* are also called the *independent* and the *objectives* are also called the *dependent* of a function.

2.1 Deterministic Local Optimization

By finding the optimum in our case we want to minimize a function. Mathematically the goal is to $\min_x f(x)$, so we search for the value of x for which $f(x)$ can be minimized. Nocedal and Wright define this as

> "A point x^* is a local minimizer if there is a neighborhood N of x^* such that $f(x^*) \leq f(x)$ for all $x \in N$ [3]".

There is a broad set of deterministic local optimization algorithms. The methods shown here follow the form

$$x_{k+1} = x_k + \alpha_k p_k \tag{2.4}$$

and are so called line search algorithms. As the name already suggests, the idea is to move from an initial starting point x_0 to the minimum of the function x^*. If the problem is linear, this can be solved by just one step. Unfortunately most problems are nonlinear, because of this we need iterative methods, therefor the indices. α_k is a positive skalar called *step length* and p_k is the *search direction*. The way these values are computed differ from algorithm to algorithm and are at least loosly explained in the following.

Search direction

In most cases the search direction should be a descending direction, which can be guaranted by using $p_k^T \nabla f_k < 0$. In our cases the search direction has mostly the form

$$p_k = -B_k^{-1} \nabla f_k \tag{2.5}$$

If we want to make it simple, B_k is just an identity matrix. Therefor only gradient information is used. More sophisticated strategies use conjugate gradients, second derivatives or approximations of the second derivatives.

Step length

For the step length α_k there exist two equations called the *Wolfe conditions* which are given without proof by

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla_k^T p_k \text{ with } c_1 \in (0, 1) \tag{2.6}$$

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k \text{ with } c_2 \in (c_1, 1) \tag{2.7}$$

Equation 2.6 is also known as *Armijo condition* whereas equation 2.7 is called *curvature condition*. Initially for Newton and quasi Newton algorithms steep length is usually one. For gradient descent algorithms step length can differ. The constant c_1 is often taken as 10^{-4}. In the following examples we use the implemented multidimensional optimization algorithms of GSL, where step length computation is included.

The methods to derive the step length and the search direction vary in the different algorithms, which are introduced in the following subsection. We just focus on the most common algorithms. Most of them exist in many slightly different styles, so a general view on the algorithms behavior is chosen. Four algorithms which are available in the GNU Scientific Library are used for testing and comparison, these are

- Steepest Descent Method, also known as Gradient Descent Method
- Conjugate Gradient Method, Fletcher-Reeves
- Conjugate Gradient Method, Polak-Ribiere
- Quasi Newton Method, BFGS

To ensure completeness, in this introductory section also Newton Method is presented. Later on it will be neglected, mainly because of the easy access of the GSL algorithms. The algorithms are now applied to a two-dimensional problem, to compare their performance and get a better feeling of their behaviour. The two dimensional function 2.8 [2] is given by

$$f(x_1, x_2) = 2x_1^4 + x_2^4 - 2x_1^2 - 2x_2^2 + 4\sin(x_1 x_2) + 5 \tag{2.8}$$

with its partial derivatives

$$\frac{df}{dx_1} = 4x_2 \cos(x_1 x_2) + 8x_1^3 - 4x_1 \tag{2.9}$$

$$\frac{df}{dx_2} = 4x_1 \cos(x_1 x_2) + 4x_2^3 - 4x_2 \tag{2.10}$$

Figure 2.2 shows the behaviour of this function in space.

[2] example function taken from Schröder, page 347-349

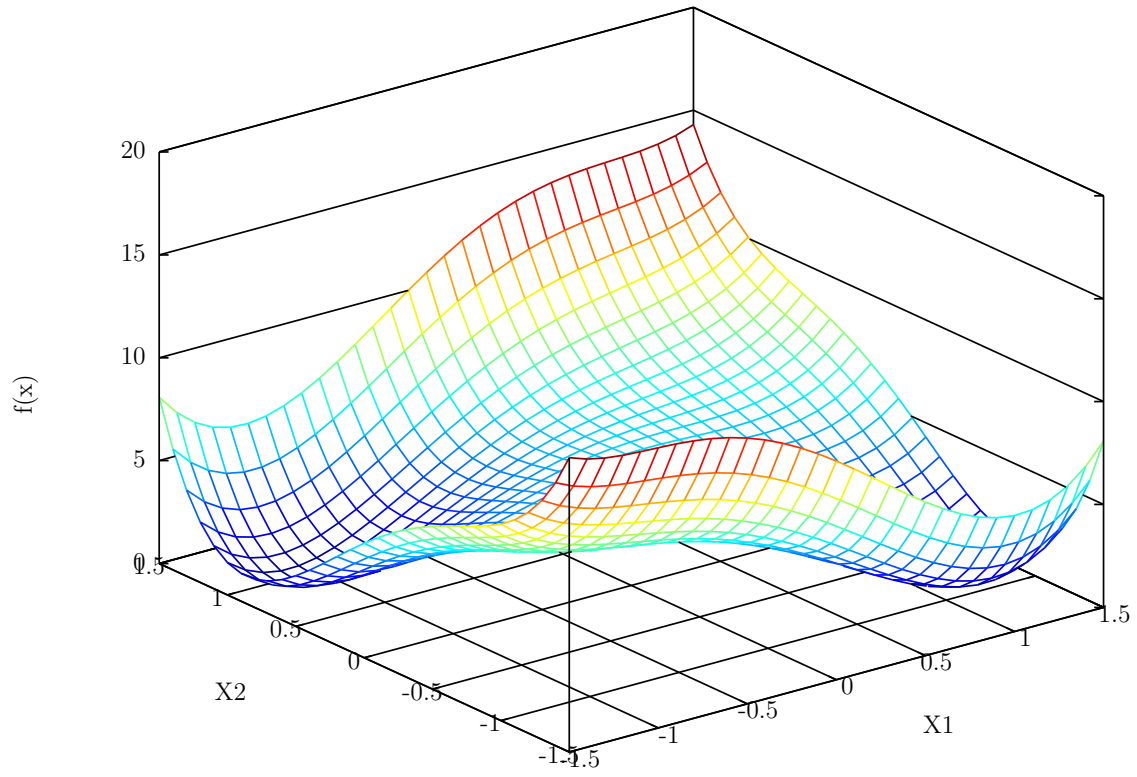


Figure 2.2 – Test function 2.8 for line search algorithms

2.1.1 Gradient Descent Method

The Gradient Descent Method (GDM) is probably one of the most basic line search methods for optimization. $B_k$is chosen as an identity matrix, so the basic formula can be simplified as

$$x_{n+1} = x_n - \alpha_n \nabla f(x) \tag{2.11}$$

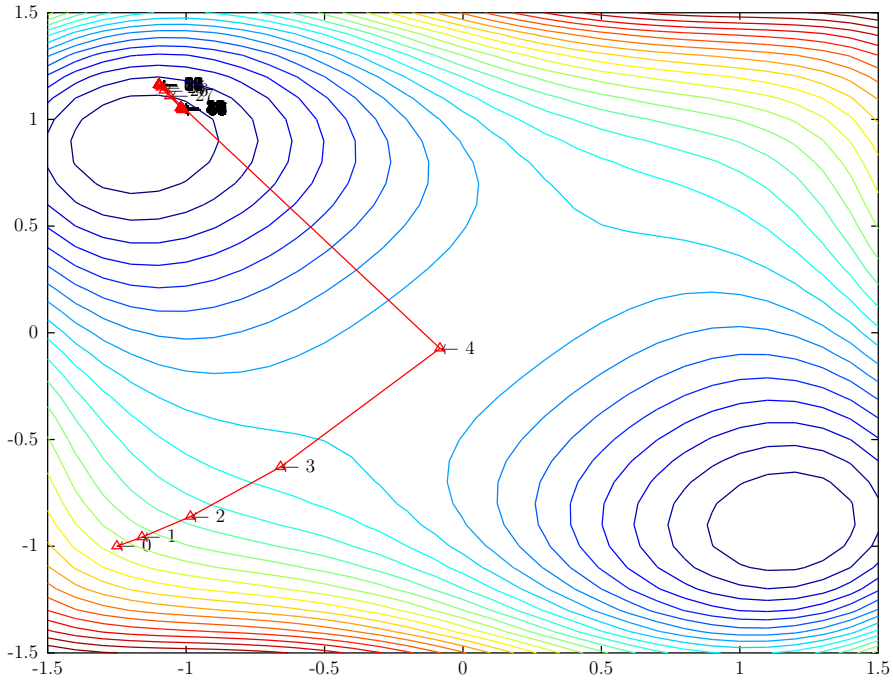


Figure 2.3 – GDM applied to function 2.8

The idea behind this is quite simple. If the multivariate function $f(x_n)$ is defined and differentiable in a neighborhood of a point a, then $f(x_n)$ decreases fastest if one goes from a in the direction of the negative gradient of f at a.

Figure 2.3 gives a detailed overview of the iterative optimization process, starting at $x_1 = -1.25$, $x_2 = -1$. After 78 iterations with an initial step length of $\alpha = 0.1$ the algorithm stops because no more progress can be reached. The iteration process is given in table 2.1.

iteration	x_1	x_2	$f(x_1, x_2)$
0	-1.25	-1	
1	-1.15947	-0.95753	8.51601
2	-0.98431	-0.86099	7.00505
3	-0.65888	-0.62840	5.48410
4	-0.08304	-0.07306	4.99992
⋮	⋮	⋮	⋮
75	-1.01211	1.04535	0.57300
76	-1.01183	1.04497	0.57300
77	-1.01127	1.04421	0.57300
78	-1.01127	1.04421	0.57300

Table 2.1 – Iteration process for minimizing function 2.8 with GDM

2.1.2 Conjugate Gradient Method

Conjugate Gradient Methods (CGM) can be used for solving iterativly large *linear* systems of equations but also for solving *nonlinear* optimization problems [3]. In the first part of this subsection we derive the method initially suggested by Hestenes and Stiefel [1] for solving linear systems with positive definite coefficient matrices. In the second subsection, we focus on two optimization algorithms for nonlinear problems which were introduced by Fletcher and Reeves [2] and by Polak and Ribiere. Both algorithms perform almost similarly.

Linear Conjugate Gradient Method

CGM can be used for solving linear systems of equations and for optimization. This means we can solve a system like equation 2.12

$$Ax = b \tag{2.12}$$

with A as an $n \times n$ matrix. This can be easily reformulated as an optimization problem like equation 2.13 does.

$$\min \phi(x) \stackrel{def}{=} \frac{1}{2} x^T A x - b^T x \tag{2.13}$$

According to Nocedal the gradient of ϕ equals the residual of the linear system [3], that is

$$\nabla \phi(x) = Ax - b \stackrel{def}{=} r(x) \tag{2.14}$$

which leads to

$$r_k = Ax_k - b \tag{2.15}$$

The basic CG algorithm can be abstracted like the following[3]:
Algorithm CG
Given x_0;
Set $r_0 = Ax_0 - b$, $p_0 = -r_0$, $k = 0$
while $r_k \neq 0$
$p_0 = -r_0, k_0 = 0.$

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k} \tag{2.16}$$

$$x_{k+1} = x_k + \alpha_k p_k \tag{2.17}$$

$$r_{k+1} = r_k + \alpha_k A p_k \tag{2.18}$$

$$\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \tag{2.19}$$

$$p_{k+1} = -r_{k+1} + \beta_{k+1} p_k \tag{2.20}$$

[3] like Nocedal and Wright, p. 112

$$k = k + 1 \tag{2.21}$$

endwhile

Non-Linear Conjugate Gradient Method - Fletcher-Reeves

As shown in equation 2.13 CGM can be used for solving optimization problems. Fletcher and Reeves (FR) were the first who showed how to solve the nonlinear problem by applying the following algorithm[4].

Algorithm FR
Given x_0;
Evaluate $f_0 = f(x_0), \nabla f_0 = \nabla f(x_0)$;
Set $p_0 = -\nabla f_0$, $k = 0$;
while $\nabla f_k \neq 0$
Compute α_k. and set $x_{k+1} = x_k + \alpha_k p_k$;
Evaluate ∇f_{k+1};

$$\beta_{k+1}^{FR} = \frac{\nabla f_{k+1}^T \nabla f_{k+1}}{\nabla f_k^T \nabla f_k} \tag{2.22}$$

$$p_{k+1} = -\nabla f_{k+1} + \beta_{k+1}^{FR} p_k \tag{2.23}$$

$$k = k + 1 \tag{2.24}$$

endwhile
Applying the FR algorithm to our previously stated problem leads to the results shown in table 2.2. In figure 2.4 we see that the number of iterations has decreased to 10 in comparison with GDM with 78 iterations. This is also shown by table 2.2. The more interesting aspect is that the FR algorithm converges to a different local minimum from that by the GDM, where the dependent function value is bigger then the previous one.

[4]like Nocedal and Wright, p. 121

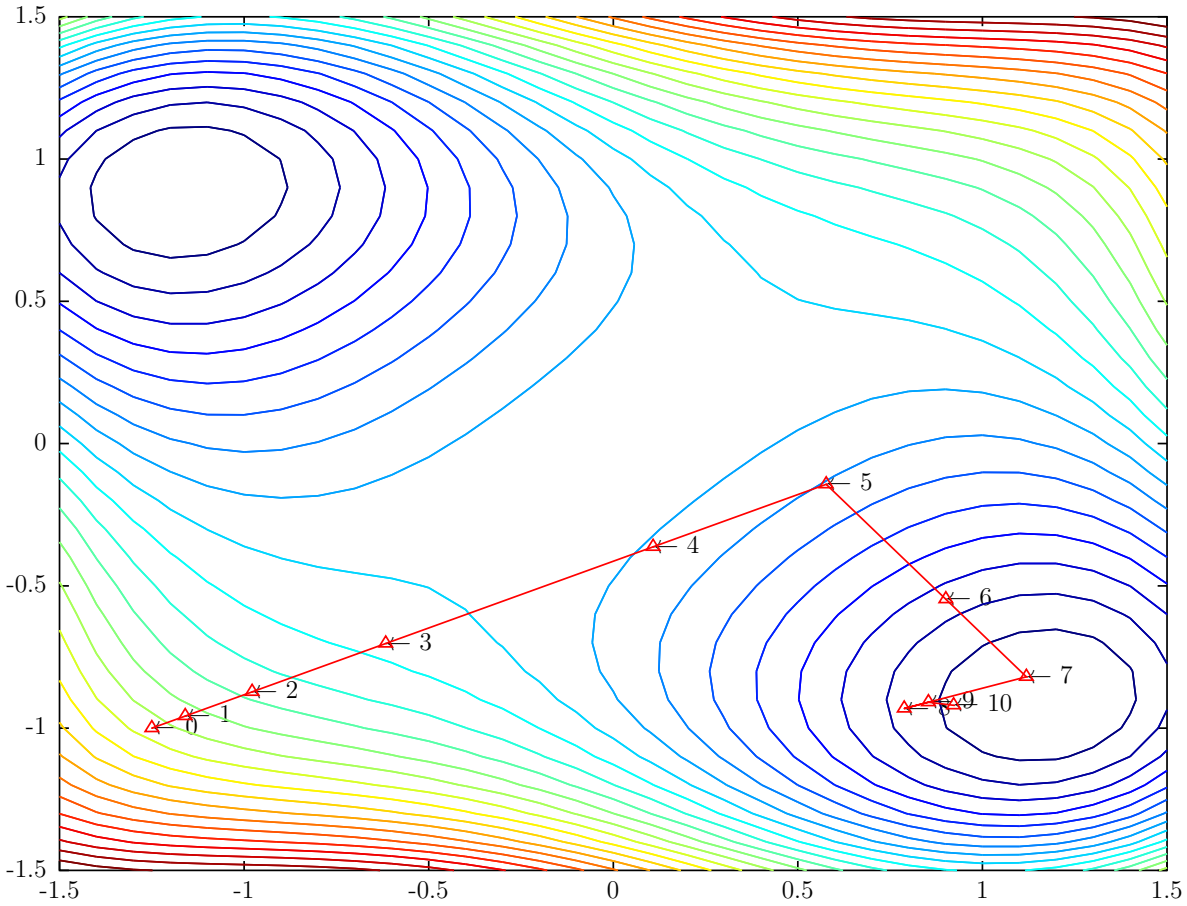


Figure 2.4 – Fletcher Reeves Method

iteration	x_1	x_2	$f(x_1, x_2)$
0	-1.25	-1	
1	-1.15947	-0.95753	8.51601
2	-0.97841	-0.87258	6.99010
3	-0.61628	-0.70268	5.46373
4	0.10797	-0.36289	4.57423
5	0.57684	-0.14292	4.18614
6	0.90067	-0.54709	2.29249
7	1.11902	-0.81960	1.56387
8	0.78807	-0.93101	0.86857
9	0.85365	-0.90894	0.83338
10	0.92196	-0.91996	0.76836

Table 2.2 – Iteration process for minimizing f(x) 2.8 with FR algorithm

Non-Linear Conjugate Gradient Method - Polak-Ribiere

The algorithm suggested by Polak-Ribiere (PR) is almost equal to the one of FR, at least both belong to the family of CGM. The main difference is how the parameter β_k is computed. In contrast to equation 2.22 the formula is

$$\beta_{k+1}^{PR} = \frac{\nabla f_{k+1}^T(\nabla f_{k+1} - \nabla f_k)}{||\nabla f_k||^2} \tag{2.25}$$

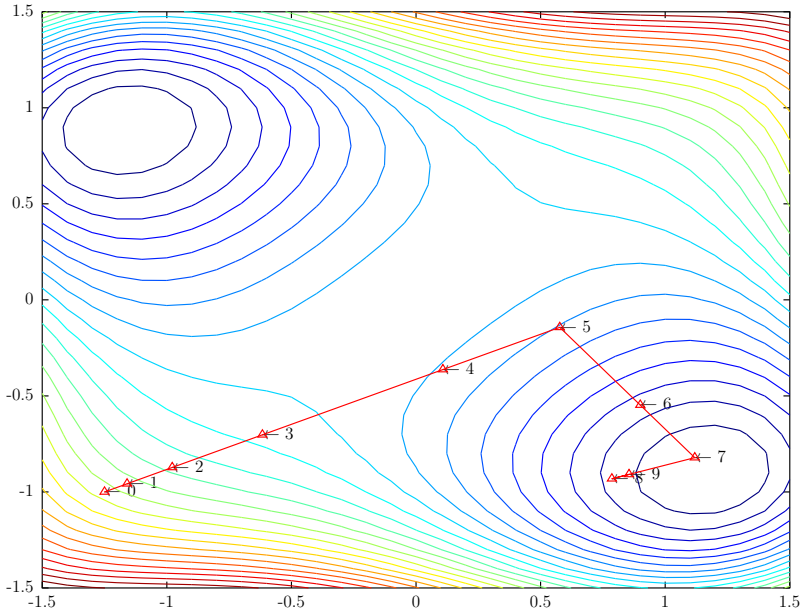


Figure 2.5 – PR Method

iteration	x_1	x_2	$f(x_1, x_2)$
0	-1.25	-1	
1	-1.15947	-0.95753	8.51601
2	-0.97841	-0.87258	6.99010
3	-0.61628	-0.70268	5.46373
4	0.10797	-0.36289	4.57423
5	0.57684	-0.14292	4.18614
6	0.90007	-0.54757	2.29004
7	1.11884	-0.82145	1.55603
8	0.78603	-0.93149	0.87131
9	0.85476	-0.90877	0.83275

Table 2.3 – Iteration process for minimizing f(x) 2.8 with PR Method

2.1.3 BFGS

In quasi-Newton methods, $B_k$is an approximation to the Hessian that is updated at every iteration by means of a low rank formula. The most popular in this family is the BFGS method, named for its discoverers Broyden, Fletcher, Goldfarb, and Shanno [3]. Without going into detail the basic algorithm is given with[5]

Algorithm BFGS
Given starting point x_0, convergence tolerance $\epsilon > 0$, inverse Hessian approximation H_0;
$k = 0$;
while $||\nabla f_k|| > \epsilon$
Compute search direction
$p_k = -H_k \nabla f_k$;
Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search procedure to satisfy the Wolfe conditions;
Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;
Compute H_{k+1} by means of;
$k = k + 1$;
endwhile
The formula for the approximation of the Hessian is

$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T \tag{2.26}$$

with

$$\rho_k = \frac{1}{y_k^T s_k} \tag{2.27}$$

and

$$s_k = x_{k+1} - x_k \tag{2.28}$$

$$y_k = \nabla f_{k+1} - \nabla f_k \tag{2.29}$$

Applying this algorithm to our problem gives fast convergence, but of course it can't tell that in general this approach is superior to the previous mentioned algorithms.

[5]like Nocedal and Wright, p.140

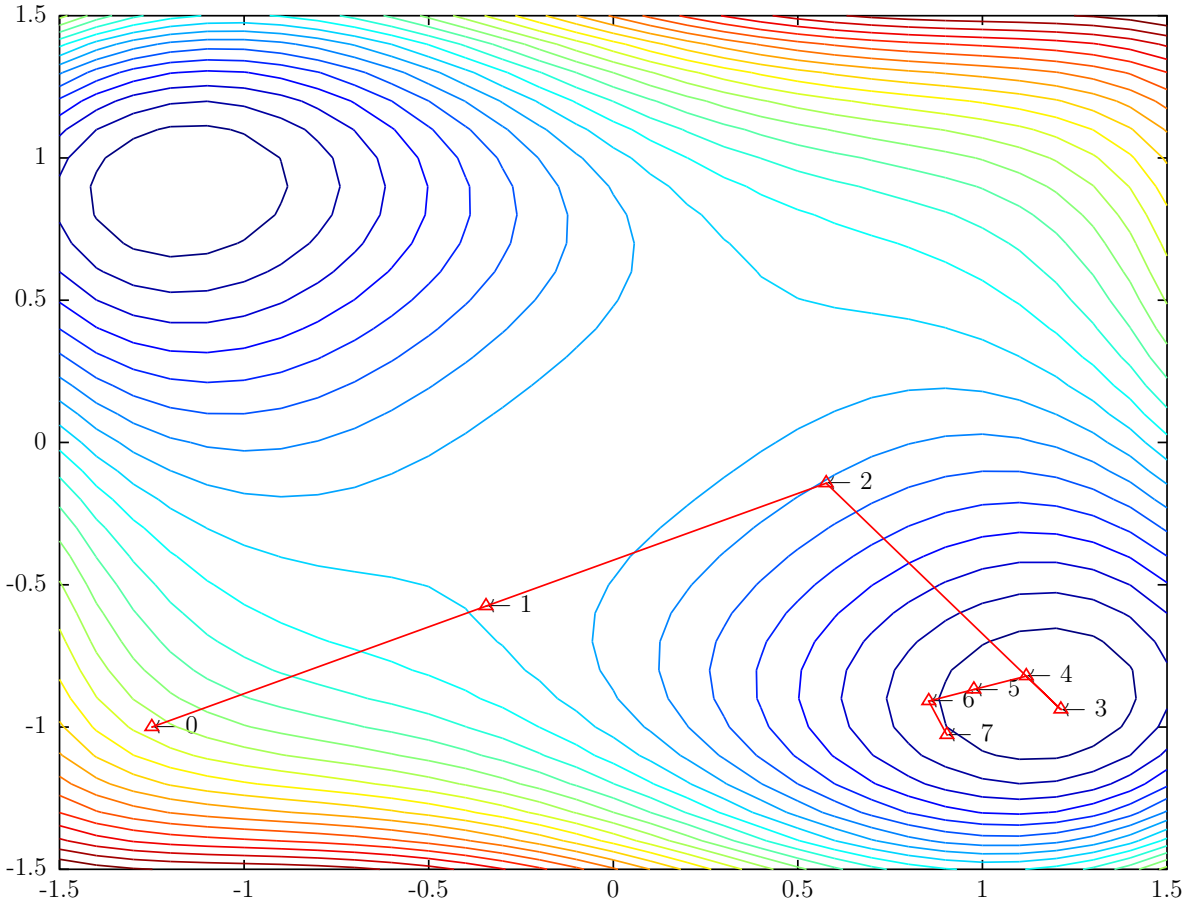


Figure 2.6 – BFGS Method for minimizing f(x) 2.8

iteration	x_1	x_2	$f(x_1, x_2)$
0	-1.25	-1	
1	-0.34468	-0.57526	5.02622
2	0.57684	-0.14291	4.18614
3	1.21238	-0.93804	1.76576
4	1.11888	-0.82106	1.55770
5	0.97675	-0.86823	0.97295
6	0.85452	-0.90880	0.83289
7	0.90372	-1.02782	0.50004

Table 2.4 – Iteration process for minimizing f(x) 2.8 with BFGS Method

2.1.4 Newton Method

Probably the most widely known method for optimization is the Newton method. The matrix B_k is constructed by taking the exact Hessian $\nabla^2 f(x_k)$. Without going to much into detail two quite similar applications of Newton method are presented here.

Newton-Raphson Method

The Newton-Rhapson (NR) method is well known for finding the root of a function by an iterative process, formulated as

$$x_{n+1} = x_n - \alpha_n \frac{f(x)}{f'(x)} \tag{2.30}$$

Let's assume a simple problem, like Papula suggested [5], with it's given derivatives[6]

$$f(x) = 2.2x^3 - 7.854x^2 + 6.23x - 22.2411 \tag{2.31}$$

$$f'(x) = 6.6x^2 - 15.708x + 6.23 \tag{2.32}$$

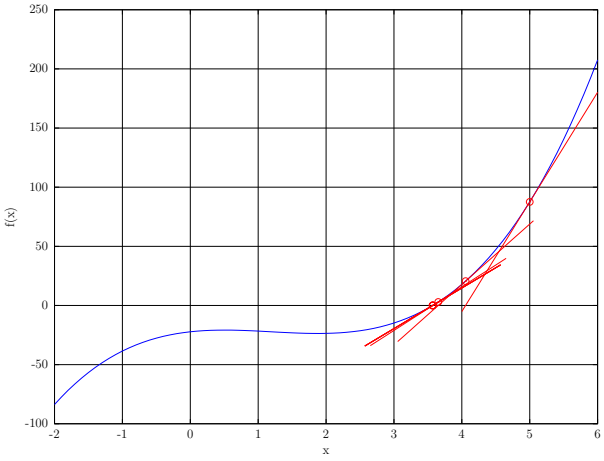


Figure 2.7 – NR for root finding in f(x) 2.31

In our example we use the starting value $x_0 = 5$, then x_1 is the cutting point of the tangent to the point $P(x_0, f(x_0))$ with the x-axes. Under certain conditions x_{n+1} lies closer to the root of f. The process can then be iterated until the change in x is lower than a given value or equal to zero. Figure 2.7 gives a geometrical interpretation of the described method. The blue curve is the graph of the function f. The red points are located at the coordinates $(x_n|f(x_n))$. The red lines are the tangents of f at the points $(x_n|f(x_n))$. The step length in this example is constant so that $\alpha = 1$.

x_n	**value**
x_0	5
x_1	4.0554
x_2	3.6523
x_3	3.5730
x_4	3.5700
x_5	3.5700

Table 2.5 – finding root of f(x) 2.31

[6] example function taken from Papula, page 388

Newton Method for Optimization

Newton Method (NM) shouldn't be mixed up with the previous introduced NR Method. The formula is just slightly different from NR Method. But instead of finding the root of a function, NM converges to the place where $f'(x) = 0$, a so called saddle point of $f(x)$. A saddle point can represent a minimum or maximum of a function. This means that NM fits well for local optimization tasks. The formula is

$$x_{n+1} = x_n - \alpha_n \frac{f'(x)}{f''(x)} \tag{2.33}$$

Now we have to enhance the previous example with the second derivative given by

$$f''(x) = 13.2x - 15.708 \tag{2.34}$$

x_n	**value**
x_0	-1
x_1	-0.012799
x_2	0.392325
x_3	0.495202
x_4	0.502818
x_5	0.502861
x_6	0.502861

Table 2.6 – finding local maximum of f(x) 2.31

x_n	**value**
x_0	4
x_1	2.6790
x_2	2.0931
x_3	1.9030
x_4	1.8776
x_5	1.8771
x_6	1.8771

Table 2.7 – finding local minimum of f(x) 2.31

The step length α_n is simply taken as 1. The behavior of this method gets clearer by figure 2.8. We use the same problem like in the previous section. Depending on the starting point a local minimum or a local maximum of f is found. With the starting point $x_0 = -1$ the method converges to a local maximum (green points, green tangents), with the starting point $x_0 = 4$ the same method converges to a local minimum (red points, red tangents).

This example shows the strong impact of the initialization value x_0 for the result of the algorithm. The example is one dimensional. Later on multidimensional problems are used, therefor we generalize Newton method. Instead of the derivative $f'(x)$ we use the gradient, $\nabla f(x)$. The reciprocal of the second derivative $f''(x)$ is replaced with the inverse of the Hessian $\nabla^2 f(x)$. So the formula is

$$x_{n+1} = x_n - \alpha_n [\nabla^2 f(x)]^{-1} \nabla f(x) \tag{2.35}$$

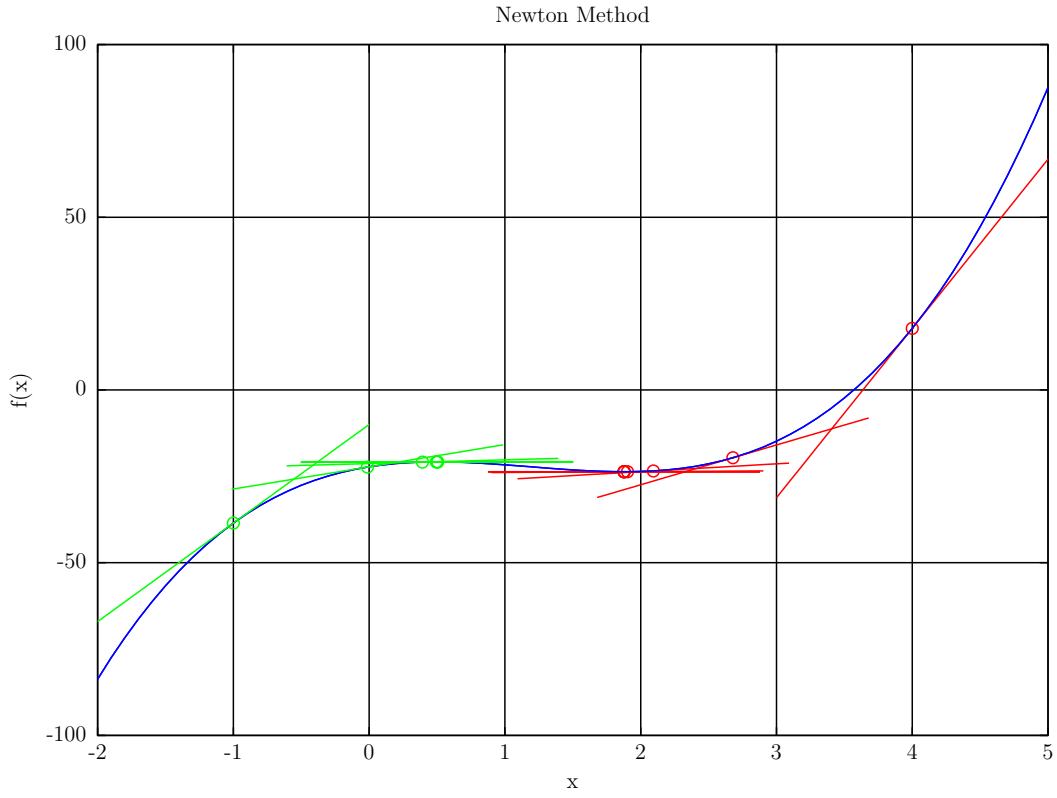


Figure 2.8 – NM for finding local maximum (green) and minimum (red) in f(x) 2.31

The calculation is shown by a simple example. Let's consider a multidimensional problem like

$$f(x) = \sum_{i=1}^{3} i x_i^2 \tag{2.36}$$

Figure 2.9 gives a two dimensional overview of the function behavior which has it's minimum at $x_1 = x_2 = x_3 = 0$. The correct formulation is

$$\begin{bmatrix} x_{1,n+1} \\ x_{2,n+1} \\ x_{3,n+1} \end{bmatrix} = \begin{bmatrix} x_{1,n} \\ x_{2,n} \\ x_{3,n} \end{bmatrix} - \alpha_n \begin{bmatrix} \frac{\delta^2 f}{\delta x_1 \delta x_1} & \frac{\delta^2 f}{\delta x_1 \delta x_2} & \frac{\delta^2 f}{\delta x_1 \delta x_3} \\ \frac{\delta^2 f}{\delta x_2 \delta x_1} & \frac{\delta^2 f}{\delta x_2 \delta x_2} & \frac{\delta^2 f}{\delta x_2 \delta x_3} \\ \frac{\delta^2 f}{\delta x_3 \delta x_1} & \frac{\delta^2 f}{\delta x_3 \delta x_2} & \frac{\delta^2 f}{\delta x_3 \delta x_3} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\delta f}{\delta x_1} \\ \frac{\delta f}{\delta x_2} \\ \frac{\delta f}{\delta x_3} \end{bmatrix} \tag{2.37}$$

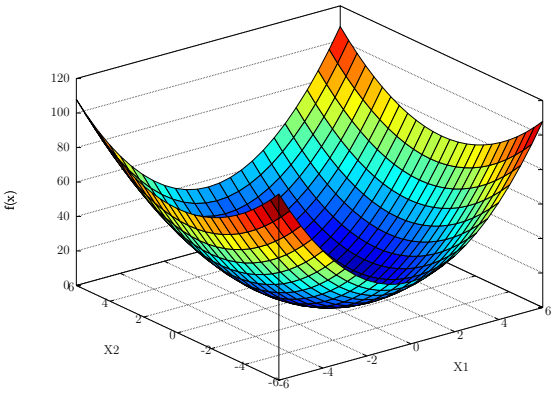


Figure 2.9 – quadratic function 2.36

I For the step length α_n we take the value 0.5, the algorithm starts with $x_1 = x_2 = 5$. If we would have taken a step length of one, then the algorithm would have found the minimum in one iteration. This is true for all quadratic problems. The step length of 0.5 is just taken for illustration purposes, as we see in figure 2.10.

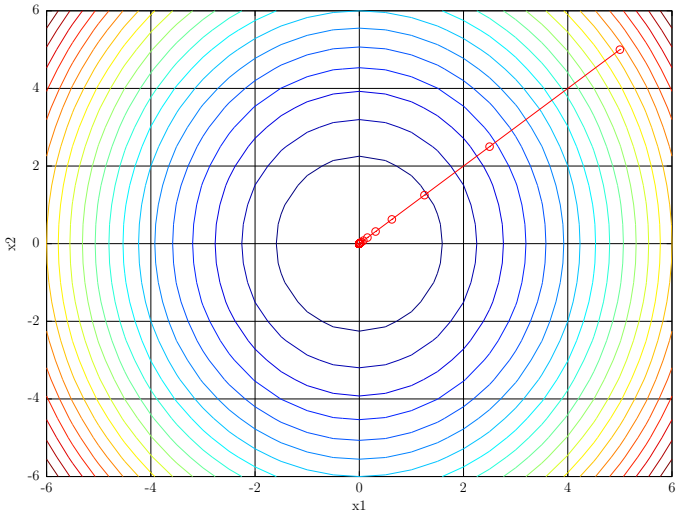


Figure 2.10 – NM for finding minimum in f(x) 2.36

2.2 Heuristic Global Optimization

Evolutionary Algorithms (EA) are very powerful search and optimization methods in the family of heuristic optimization methods. The optimal solution is found by searching through a population of different feasible solutions. After the population is studied in each iteration, the elicits are selected and are moved to the next generation applying genetic operators. After a sufficient number of generations, better solutions dominate the search space therefore the population converges towards the optimal solution [14]. This statement of Bagheri and Deldari describes the idea of evolutionary algorithms quite well.

But in contrast to the deterministic algorithms introduced in the last section, which often get trapped in local minimas due to the initial value of the independents, these algorithms converge mostly to the global minimum. Another difference between deterministic and evolutionary algorithms is that evolutionary algorithms in general don't require derivatives. Of course there are also deterministic algorithms which don't need derivatives, nevertheless this is something worthwhile to mention.

There are plenty of questions which have to be cleared before a reasonable algorithm setup is developed. First of all the optimization problem has to be stated in a way that it satisfies the following definition:

The optimization problem has to be defined by

- a search space Ω,
- an objective function $f : \Omega \to \mathbb{R}$ which relates every possible solution a function value
- and a comparison relation $\succ \in \{<, >\}$.

Then the global optima $X \subseteq \Omega$ is defined as [6]:

$$X = \{x \in \Omega | \vee x' \in \Omega : f(x) \preceq f(x')\} \tag{2.38}$$

The next step is about defining the evolution process. In the most general terms, evolution can be described as a two-step iterative process, consisting of random variation followed by selection [7]. These two main building blocks variation and selection can be implemented in various ways. Furthermore the detail operators can be implemented in different ways. A detailed description on the single classes and methods of the core program is given in the following sections. Figure 2.11 gives a general overview of the evolutionary algorithm framework [24].

2.2.1 Population, Chromosomes, Genes

The starting point of every EA is the initial population. A population consists of members, which are often called chromosomes. Furthermore the chromosomes consist of a number of gene strings.

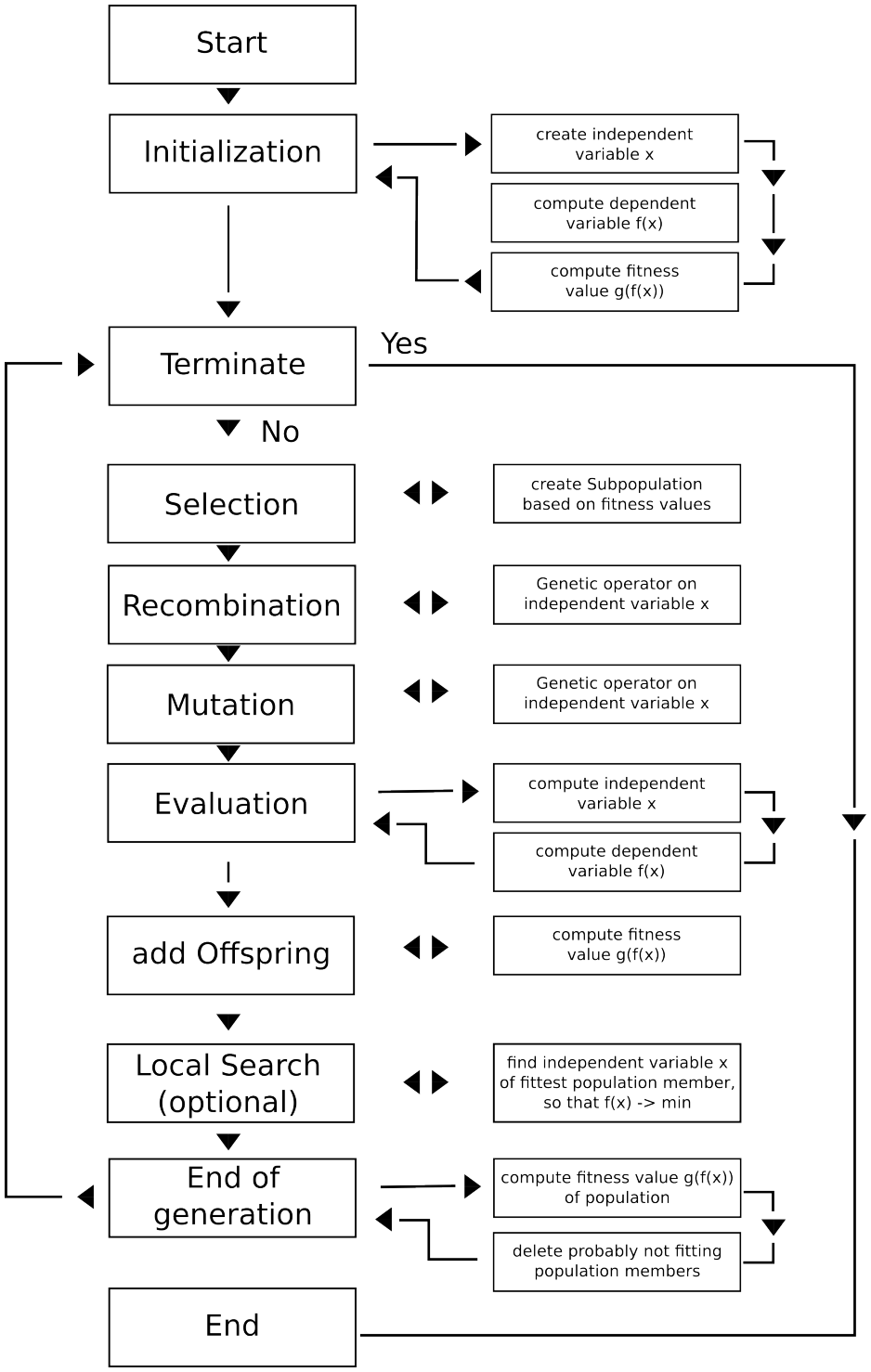


Figure 2.11 – The general EA framework

Each chromosome with its genes represents a possible solution to the optimization problem. The program implemented in this work follows the object oriented programming style, this means that the population, each chromosome and also each gene is an object. The link between the different classes is shown by the class diagram in figure 2.12.

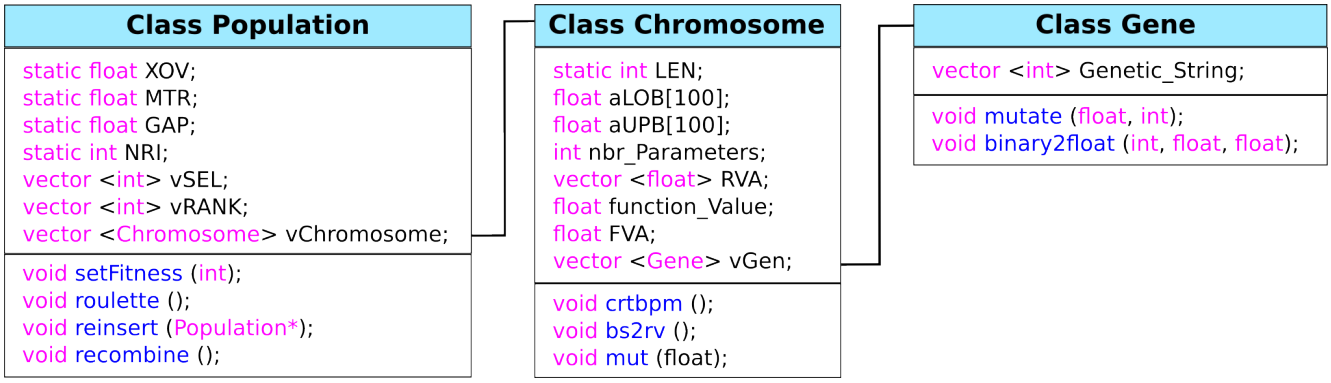


Figure 2.12 – Class diagram

The variables of the three classes are described more precisely in the following tables.

Parameter	Abbreviation	data type	Meaning
crossover rate	XOV	static float	parameter for the recombination method
mutation rate	MTR	static float	parameter for the mutation rate
generation gap	GAP	static float	parameter defines number of offspring relative to main population
nbr. of individuals	NRI	static int	number of chromosomes in the population
Chromosomes	vChromosome	vector <Chromosome>	vector including the population members
selected indexes	vSEL	vector <int>	vector used for the selection process
ranking	vRANK	vector <int>	vector used for fitness assignment

Table 2.8 – object properties of population

Parameter	Abbreviation	data type	Meaning
real value	RVA	vector <float>	the variables value as float
length	LEN	static int	length describes the number of used zeros and ones
lower boundary	LOB	static float	lower boundary of variable interval
upper boundary	UPB	static float	upper boundary of variable interval
fitness value	FVA	float	parameter descibing objects fitness in problem domain
genes	vGEN	vector <GENES>	vector containing the genes
nbr parameters	nbr_parameters	static int	number of parameters one chromosome consists of
f(x)	function_Value	float	value of chromosome in problem domain
genes	vGEN	vector <GENES>	vector containing the genes

Table 2.9 – object properties of chromosome

Parameter	Abbreviation	data type	Meaning
gen alleles	genetic_string	vector <int>	vector which contains the single values values of a gene

Table 2.10 – object properties of gene

EAs are stochastic processes. The probability values for recombination and mutation influence the population as a whole, so they are defined as static parameters of the population class. This is also true for the generation gap, which determines the size of the created sub population with offspring in each generation. Other parameters are mainly container elements.

The parameters addressing the problem domain like the boundaries and dimension of the search room are defined on the chromosome level. For the construction of a chromosome, the values LOB, UPB, nbr_parameters and LEN have to be known.

For each dimension of the search room a gene object is constructed by a random process. The single alleles of the genetic string , the zeros and ones, are stored in the genetic_string vector.

To make the linking between the classes clearer, let's assume a one dimensional problem with a search space in the interval $[0, 15]$. Then the user has to state furthermore the length of the genes, for example 8. Now a new chromosome with one gene can be constructed. The gene will look somehow like [11010000]. In combination with the chromosome parameters

- $c.LEN = 8$
- $c.LOB = 0$
- $c.UPB = 15$

the real value RVA of x can be computed. This is done by using the formula:

$$RVA = LOB + \frac{UPB - LOB}{2^{LEN} - 1} * \sum_{i=0}^{LEN-1} (GEN[i] * 2^i) \tag{2.39}$$

If we substitute the values of the example, then the real value $c.RVA = 0.647$ is computed with

$$RVA = 0 + \frac{15 - 0}{2^8 - 1} * (1 * 2^0 + 1 * 2^1 + 1 * 2^3) \tag{2.40}$$

Figure 2.13 shows the chromosome with the genetic string used in the example. By changing the interval boundaries $c.LOB$ and $c.UPB$ also $c.RVA$ would change, even if the genetic string doesn't change.

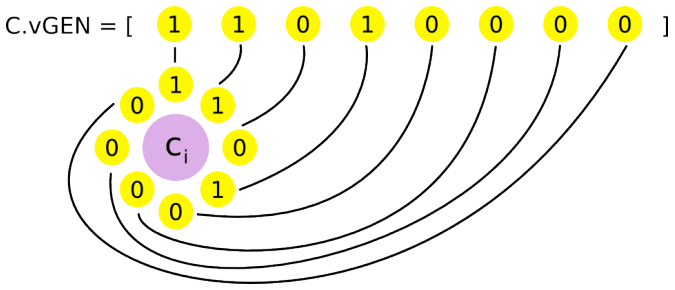


Figure 2.13 – chromosome with genetic string, stored in a vector

The resolution in the interval $[LOB; UPB]$ can be computed with

$$\frac{UPB - LOB}{2^{LEN-1}} \tag{2.41}$$

This means that the resolution behaves reciprocal to the length of the genetic string.

By using this three-level-architecture (population consisting of chromosomes consisting of genes) scalability of the program is ensured. There are no limitations on the dimensionality of the objective function. Furthermore a stated problem can be solved several times with different probability settings and population sizes. The results then can be compared with each other in order to find the best setup for a given problem.

2.2.2 Operators

The single operators are implemented as methods of the class they affect. For example the mutation operator works on the single gene, so it's implemented as a method of the gene class. In contrast the selection operator works on the whole population, so it's implemented as a method of the population class.

Fitness assignment

To assign a fitness value to a population member, two functions are of interest:

- the objective function $f(x)$, it gives a measure of how the chromosome has performed in the problem domain
- the fitness function $g(x)$, it transforms the value of the objective function into a relative value of fitness [9], thus:

$$F(x) = g(f(x)) \tag{2.42}$$

$F(x)$ must be non-negative. The fitness value of an individual has a great influence on the number of offspring it will probably produce. Whilst this fitness assignment ensures that each individual has a probability of reproducing according to its relative fitness, it fails to account for negative objective function values. A linear transformation which offsets the objective function [11] is often used prior to fitness assignment, such that

$$F(x) = a * f(x) + b \tag{2.43}$$

where a is a positive scaling factor if the optimization is maximizing and negative if we are minimizing. The offset b is used to ensure that the resulting fitness values are non-negative.

In addition to this deterministic approach fitness values also can be assigned by statistical ranking [10]. This is especially useful to limit the reproductive range, so that no individual can generate an excessive number of offspring, which would lead to an premature convergence [8]. Here, individuals are assigned a fitness value according to their rank in the population rather than their raw performance. The variable MAX is used to determine the bias, or selective pressure, towards the most fit individuals. The fitness of individuals in the population is then calculated as

$$F(x_i) = 2 - MAX + 2 * (MAX - 1)\frac{x_i - 1}{N_{ind} - 1} \tag{2.44}$$

Selection

The selection process is a stochastic process based on the fitness values of the chromosomes. The selection of individuals can be viewed as two separate processes [8]:

- determination of the number of trials an individual can expect to receive, and
- conversion of the expected number of trials into a discrete number of offspring

The first part is concerned with the transformation of raw fitness values into a real-valued expectation of an individual's probability to reproduce and is dealt with in the previous subsection as fitness assignment. The second part is the probabilistic selection of individuals for reproduction based on the fitness of individuals relative to one another and is sometimes known as sampling.

The basic concept used here is the so called "Roulette Wheel Selection". Each individual is assigned a proportion of a roulette wheel. The size of the proportion is determined by the size of the fitness value of the individual, thus individuals with high fitness values get a bigger proportion on the roulette wheel. The range of the roulette wheel is from zero to the sum of all fitness values. The selection itself is done by "turning the wheel". First a random number in the range of the roulette wheel is generated, then the corresponding individual is selected. The number of selection processes is determined by the generation gap GAP. For example, if a population consists of 100 individuals and GAP equals 0.7, then 70 selection processes are done. In each single process each individual can be selected. This also means that an individual can be selected more than once.

The probability P that an individual c of an population p with x individuals is selected is

$$P = \frac{c.FVA}{\sum_{i=1}^{x} c_i.FVA} \tag{2.45}$$

Figure 2.14 shows on the left side a population consisting of 10 chromosomes, and on the right side the roulette wheel. Each chromosome has its area on the wheel, the size of the area is determined by the chromosomes fitness value. For a GAP of 0.4, 4 chromosomes are selected.

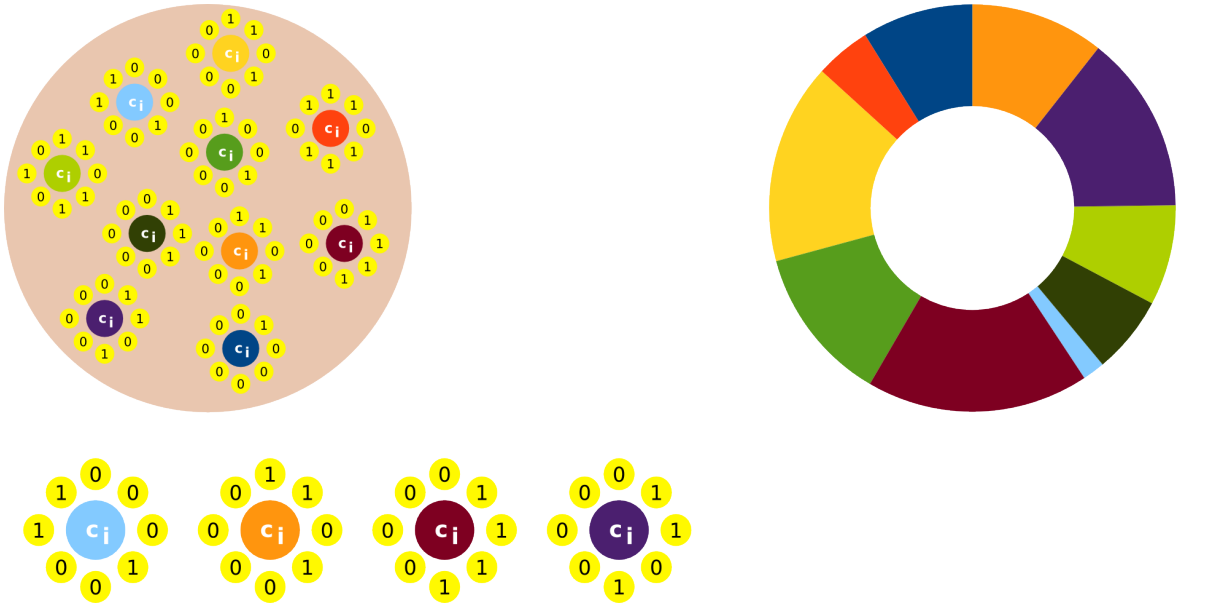


Figure 2.14 – roulette wheel selection

Recombination

Recombination is the basic operator to "produce" new chromosomes. The connection to biology is quite strong. The genetic material of two existing chromosomes is recombined, like in nature where two individuals offspring carries the genetic material of both parents. The operator itself is designed quite simple. Single-point crossover is used, this means that the genetic strings are cut in two parts at one point and then are recombined with each other. As crossing point the mid-position of the gene string is used. The figure 2.15 illustrates the recombination operator. The chromosomes c_i and c_j are recombined with each other.

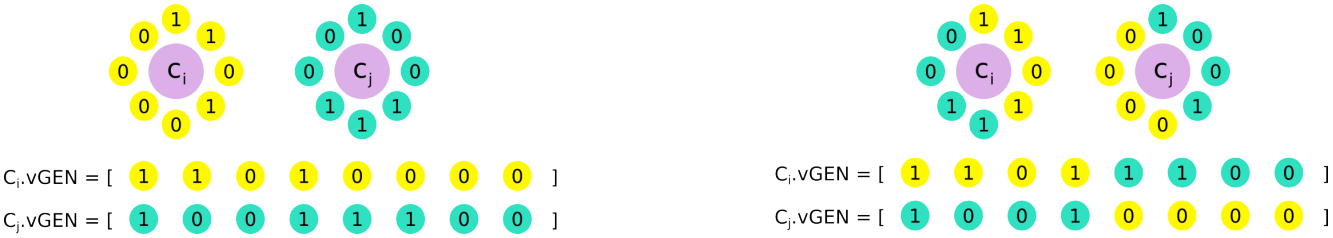


Figure 2.15 – recombination of two chromosomes

Mutation

Mutation is the second operator besides of recombination which helps to explore search space. In natural evolution, mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure [8]. In Genetic Algorithms, mutation is randomly applied with low probability, typically in the range 0.001 and 0.01, and modifies single elements in the genetic

string. Another more general way to compute mutation probability P is

$$P = \frac{0.7}{vGen.length()} \tag{2.46}$$

$vGen.length()$ is the length of the gen string. This value is selected as it implies that the probability of any one element of a chromosome being mutated is approximately 0.5 [11]. Usually considered as a background operator, the role of mutation is often seen as providing a guarantee that the probability of searching any given string will never be zero and acting as a safety net to recover good genetic material that may be lost through the action of selection and crossover [11]. Figure 2.16 illustrates the mutation process. The genes of the chromosome c_i mutate at three positions. The genes are binary encoded, so mutation means change from zero to one respectively one to zero.

Figure 2.16 – mutation of a chromosome

Reinsertion

Once a new population has been produced by selection, recombination and mutation of individuals from the old population, the fitness of the individuals in the new population may be determined [8]. In the case where the number of new individuals produced at each generation is one or two, the GA is said to be steady-state [12] or incremental [13]. If one or more of the most fit individuals is deterministically allowed to propagate through successive generations then the GA is said to use an elitist strategy [8].

In figure 2.17 such a reinsertion process is shown. In the upper left panel the population is shown, in the upper right panel the created offspring off this population. In the middle the fitness values for each chromosome of the population and the offspring are illustrated with columns. Now for each offspring chromosome it is controlled whether an less fit chromosome exists in the population. If yes, then the offspring is reinserted and the replaced chromosome is deleted. The lower part of the figure shows on the left the new population, on the right the chromosomes which were replaced.

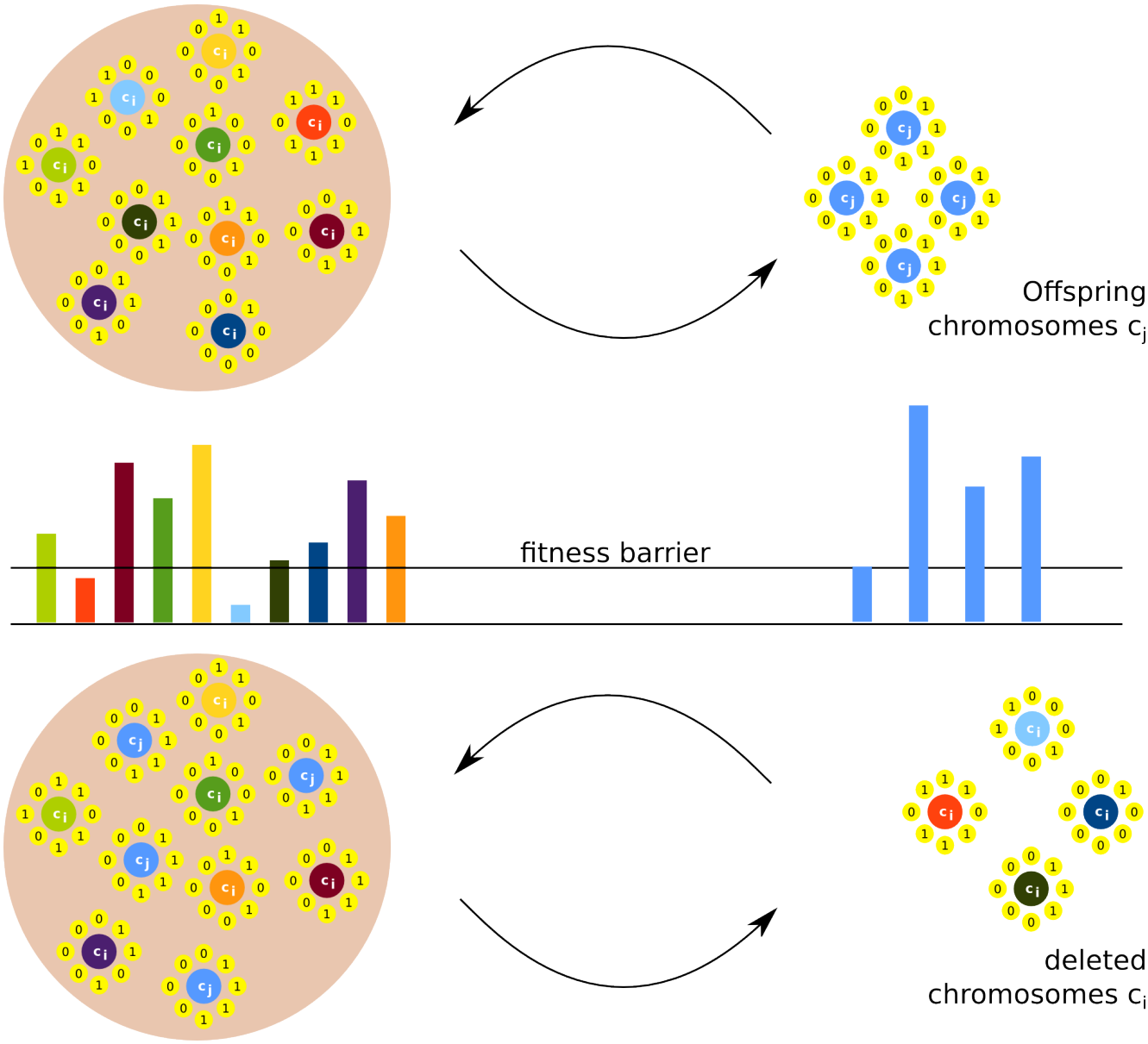


Figure 2.17 – Reinsertion process using elitist strategy

2.2.3 Parallelization

Until recent years, sequential GAs have received the greatest attention from the research community. However, parallel GAs have many interesting unique features that deserve in-depth analysis [18] [19]. These characteristics include [20]

- the reduction of the time to locate a solution (faster algorithms),
- the reduction of the number of function evaluations (cost of the search)
- the possibility of having larger populations thanks to the parallel platforms used for running the algorithms, and
- the improved quality of the solutions worked out.

For the algorithm developed in this work thread-level parallelism is included with an steering mechanism to avoid a thread overhead on the physical available cores. The focus lies on run time reduction of the algorithm. For the implementation the freely available C++ library Boost version 1.51.0 is used [7].

The algorithm calls the objective function for each chromosome in parallel. This means that when for the population members the objective function is called, depending on the number of available cores of the CPU multiple threads are added to a thread group. Each thread calculates for one chromosome the objective function. By doing so it is ensured that always 100 % of the available CPU power is used. The reduction of runtime is almost direct proportional to the number of cores.

```
1  int x = Boost::Thread::get_nbr_of_Cores ( );
2  Boost::thread_group threadGroup;
3
4  do {
5    for ( int j = 0; j < x; j ++) {
6
7      if ( int i < vChromosome.length ( ) ) {
8
9        createThread(objectiveFunction (parameters) );
10
11     }
12
13   i++;
14
15   }
16   threadGroup.join ( );
17 }while ( i < vChromosome.length ( ) );
```

Listing 2.1 – Parallelization of objective function call using Boost

[7] available from: `www.Boost.org`

2.2.4 Applications for evolutionary algorithms

In order to evaluate the developed genetic algorithms and the influence of different parameters for mutation rate, population size and population gap, the settings are tested on a set of functions. These test functions can be described by different mathematical properties such as (for completeness the definitions are stated):

- continuous / non-continuous

  According to [16] by using the Epsilon-Delta-criterion it can be said that the function $f : D \to \mathbb{R}$ is steady in $\xi \in D$ if in each $\epsilon > 0$ a $\delta > 0$ exists so that for all $x \in D$ with $|x - \xi| < delta$ it is true:

  $$|f(x) - f(\xi)| < \epsilon \tag{2.47}$$

- convex / non-convex

  a function $f : D \to \mathbb{R}$ defined on an interval $x_1 \leq x \leq x_2$ is called convex if the graph of the function lies below the line segment joining any two points of the graph [25].

- unimodal / multimodal

  a function $f : D \to \mathbb{R}$ is a unimodal function if for some value m, it is monotonically increasing for $x \leq m$ and monotonically decreasing for $x \geq m$. In that case, the maximum value of $f(x)$ is $f(m)$ and there are no other local maxima [26].

- quadratic / non-quadratic

  According to [27] a quadratic function is a polynomial function of the form

  $$f(x) = ax^2 + bx + c \tag{2.48}$$

- low dimensional / high dimensional
- with or without Gaussian noise

The idea of this different test functions was originally stated by De Jong [9]. But instead of copying the five functions introduced by De Jong, an extended set of test functions is used. De Jong introduced five function, from which the function one to four is used. Then as enhancement Rastrigin function and Goldstein & Price function are added. For finalizing the task the last two problems are

- a function inversion problem, which is described by Reed and Marks [15]
- a function inversion problem targeting the retrieval of cloud parameters

The last task is done with one, two, and three unknown. This example is especially interesting because in contrast to the previous examples it solves a real world problem instead of a theoretical one.

The testing of the Evolutionary Algorithm on each of the functions is done by two steps. At first the influence of the recombination and mutation rate on the quality of the minimum estimation has to be quantified. Because we know the global minimum we can quantify the quality by means of

- mean difference between real minimum and simulated minimum
- standard deviation of the difference between real minimum simulated minimum

As mutation rate values in the range from 0.01 to 0.09 with an step length of 0.01 are used. For recombination rate the values 0.6, 0.7, 0.8 and 0.9 are used. Each possible solution of both factors is computed 1000 times. The results of this runs are stored in logfiles and visualized on graph plots. As a result of this test series the best fitting recombination and mutation rate for the given problem can be found.

Then a detailed look is taken on the behavior of the algorithm for the optimal setup which was obtained before. A table shows the chosen rates and other characteristics like population size, number of generations until convergence is assured and so on. Four more figures are included here for visualization purposes.

- one figure showing how the value of the dependent variable of the fittest population member changes over each generation - by interpreting this figure we can make a statement how fast the optimum was found
- one figure showing how the mean value of the dependent variable of the whole population member changes over each generation - this depicts how the whole simulation converges to the probable minimum
- one figure visualizing the independent values of the fittest population member over the evolution process
- one figure for the mean value of the independent variable of the whole population during the evolution process

By interpreting these four figures a deeper understanding of the algorithm behavior and the evolution process is possible. Weaknesses and uncertainties can also be determined. At least it is possible to make a statement whether the algorithm fits for the problem and how reliable the results are.

Test function one

The first test function is given by

$$f(x) = \sum_{i=1}^{3} x_i^2 \tag{2.49}$$

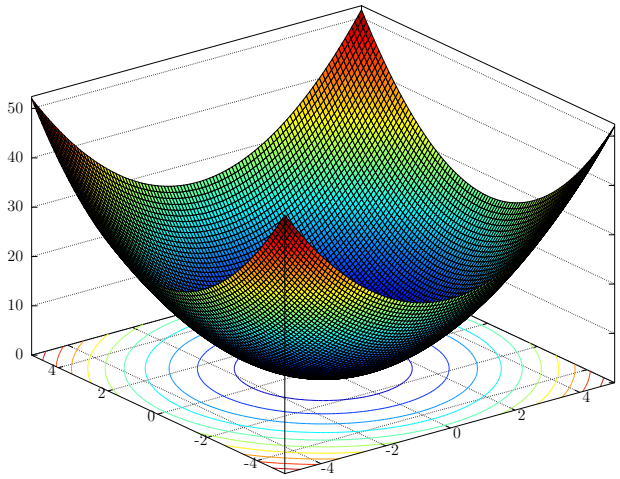
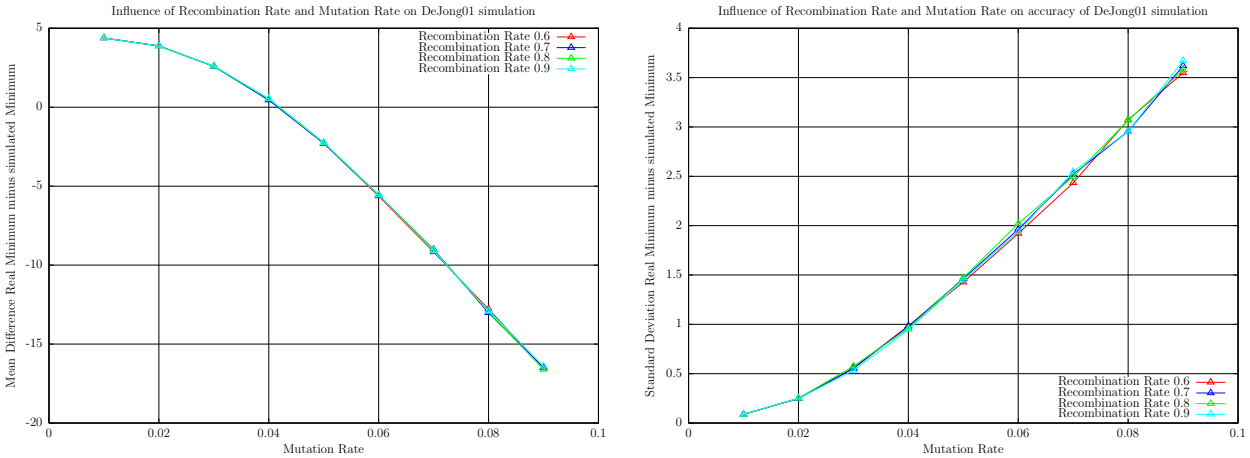


Figure 2.18 – Plot of function 2.49 in 3D - Space

It is a simple sphere which is continuous, convex, unimodal, quadratic and low dimensional. No noise is added. As seen from figure 2.18, the function fits quite well for deterministic methods like Newton. The analysis of the different combinations of recombination and mutation rates shows clearly, that for minimizing the difference between real minimum and simulated minimum a mutation rate of 0.04 has to be chosen. The influence of the recombination rate can be neglected, for the in depth analysis a value of 0.7 was chosen. As proof figure 2.19(a) and figure 2.19(b) are added. Another finding here is that a raising mutation rate also means a raising standard deviation in the estimates, what at least is plausible by the nature of the mutation operator.



(a) Mean Difference $f(x^*) - f(\hat{x})$

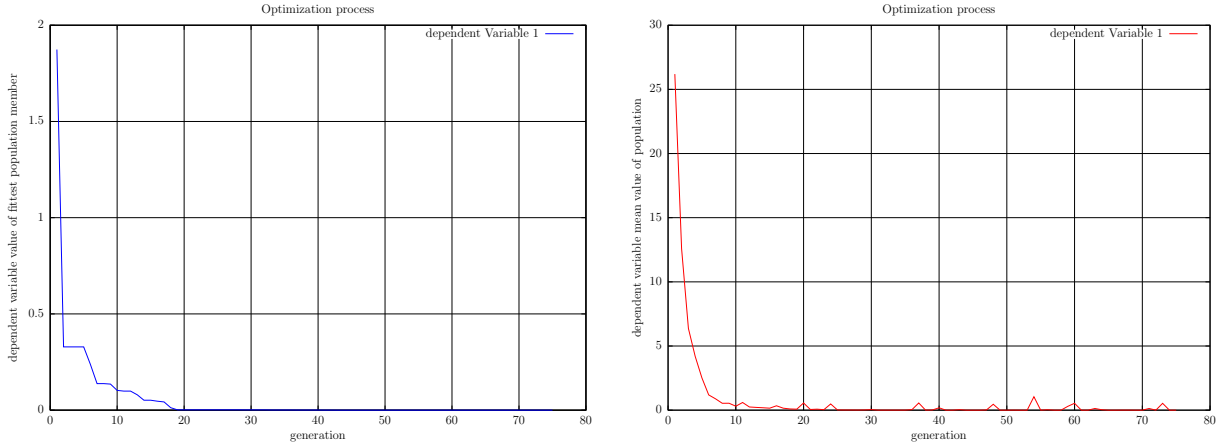
(b) Standard Deviation $f(x^*) - f(\hat{x})$

Figure 2.19 – Development of Dependent Variable Value function 2.49

Parameter	**value**
Population size	50 chromosomes
Nbr. of generations	78
Generation gap	0.85
Mutation rate	0.04
Crossover rate	0.7
gene length	11
search space	$[-5.12 : 0.01 : 5.12]$
possible solutions	$(512)^3$
MAX $f(\pm 5.12, ..., \pm 5.12)$	$= 78.6$
MIN $f(0, 0, 0)$	$= 0$

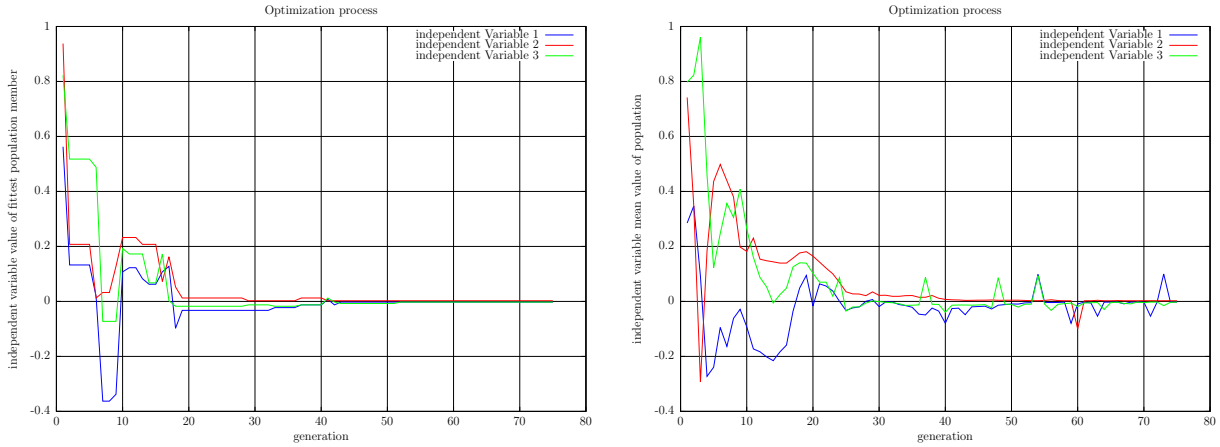
Table 2.11 – parameters space function 2.49

Table 2.11 lists the parameters which were used for the detailed view on an optimization process. The dependent variables value converges fast to the global minimum (figure 2.20(a) and figure 2.20(b)), while a nearly optimal solution is also found quite fast and not changing then any more (figure 2.21(a) and figure 2.21(b)). As stopping criterion the independent variables values were chosen. If they don't change for a period of 20 generations, the algorithm terminates. The calculated values after 78 generations of the fittest chromosome were $x_1 = -0.0025$, $x_2 = 0.0025$, $x_3 = -0.0025$, $f(x) = 0.00001877$.



(a) Fittest Population Member (b) Mean Value of Population

Figure 2.20 – Development of Dependent Variable Value function 2.49



(a) Fittest Population Member (b) Mean Value of Population

Figure 2.21 – Development of Independent Variable Values function 2.49

Rosenbrock Function

Rosenbrock function is given by

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \tag{2.50}$$

It is also called Banana function, because of the long valley which lookes like a banana, if we watch the plot 2.22 from above. The function is continuous, convex, unimodal, quadratic and low dimensional. No noise is added. This function is also solvable for deterministic methods, but it's much harder than that in the previous example. The main issue is that in the valley there are a lot of terrace points at which algorithms can get stuck.

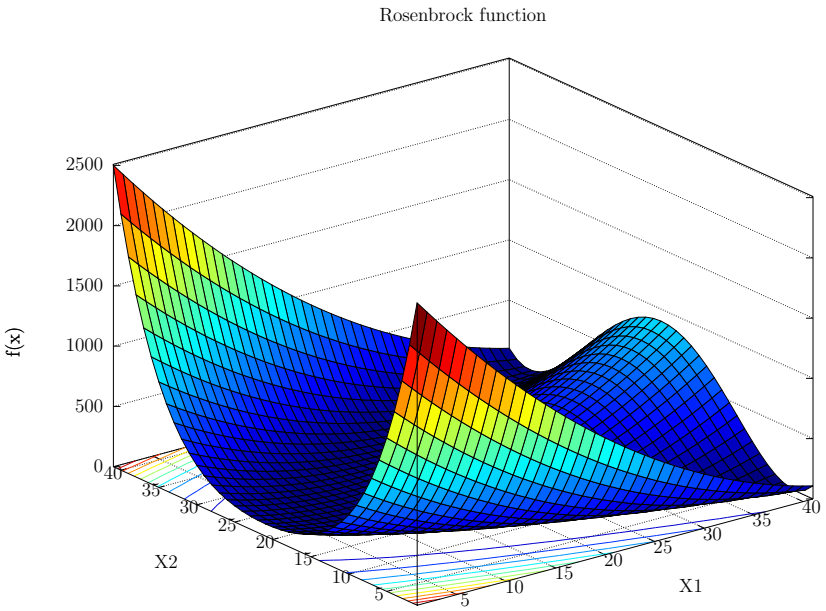


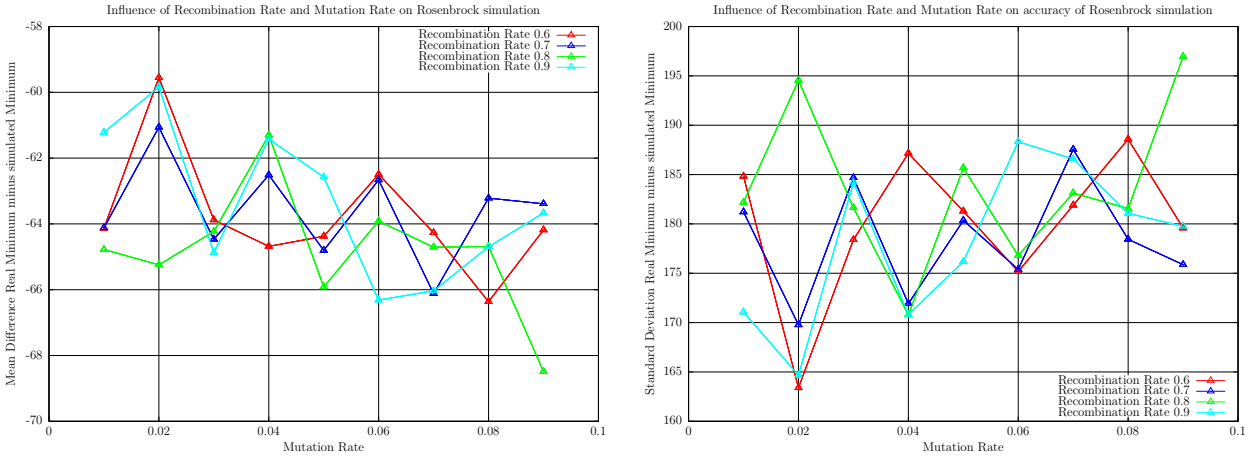
Figure 2.22 – Rosenbrock function 2.50

Figure 2.23(a) and figure 2.23(b) give an quite interesting view on the influence of different recombination and mutation rate combinations on the quality of the estimation. For finding a compromise a mutation rate of 0.02 and a recombination rate of 0.6 were chosen as the best setup for further analysis. The plots don't make it that easy than in the first De Jong function, but by taking the number of 1000 simulations for each combination into account it can be said that these results are reliable, even though we cannot explain them.

In contrast to the first test function we took here stability of 7 loops for the stopping criterion. Figure 2.24(a) and figure 2.24(b) give a quite clear picture of what happens. In figure 2.24(a) we see that a quite optimal solution is rapidly chosen to be the fittest population member and not changing any more. Figure 2.24(b) supports this . Here we see moreover, that the whole population rapidly converged down into the banana valley. But if we take also figure 2.25 and figure 2.26 into account,it can be realized that the convergence process for the whole population has not been finished yet. Allowing a stricter stopping criterion, like 20 loops for ensuring convergence, would probably lead to better results.

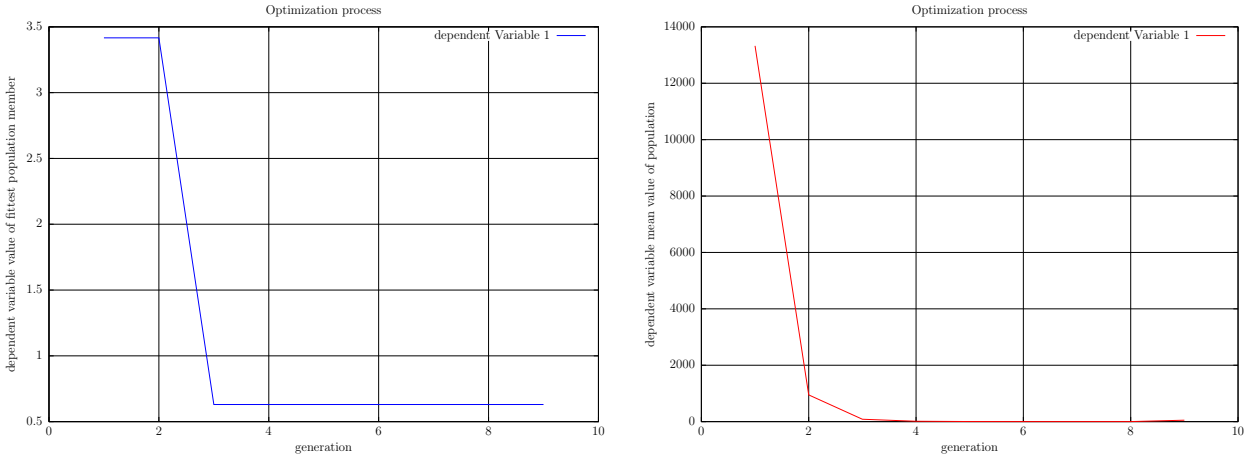
Parameter	**value**
Population size	50 chromosomes
Nbr. of generations	9
Generation gap	0.85
Mutation rate	0.02
Crossover rate	0.6
gene length	10
search space	$x_i = [-5.12 : 0.02 : 5.12]$
possible solutions	$(512)^2$
MAX $f(-5.12, 5.12)$	$= 44534$
MIN $f(1, 1)$	$= 0$

Table 2.12 – parameters for testing with function 2.50



(a) Mean Difference $f(x^*) - f(\hat{x})$ (b) Standard Deviation $f(x^*) - f(\hat{x})$

Figure 2.23 – Influence of Recombination and Mutation Rate on Estimation for f(x) 2.50



(a) Fittest Population Member (b) Mean Value of Population

Figure 2.24 – Development of Dependent Variable Value in function 2.50

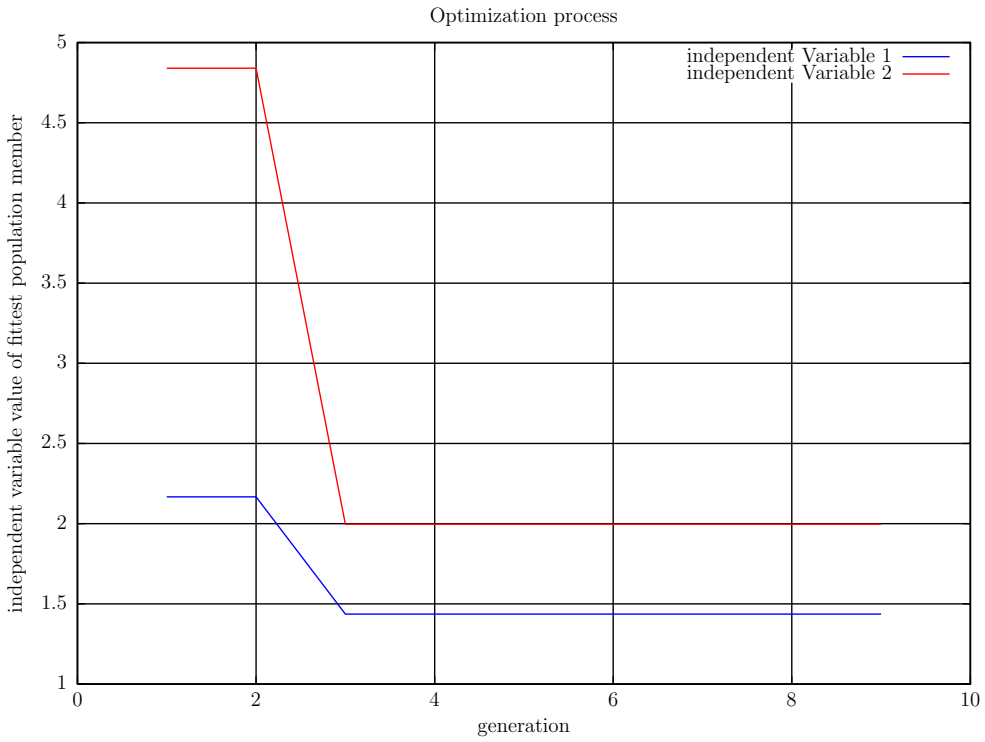


Figure 2.25 – Fittest Population Member Independent Variable Value function 2.50

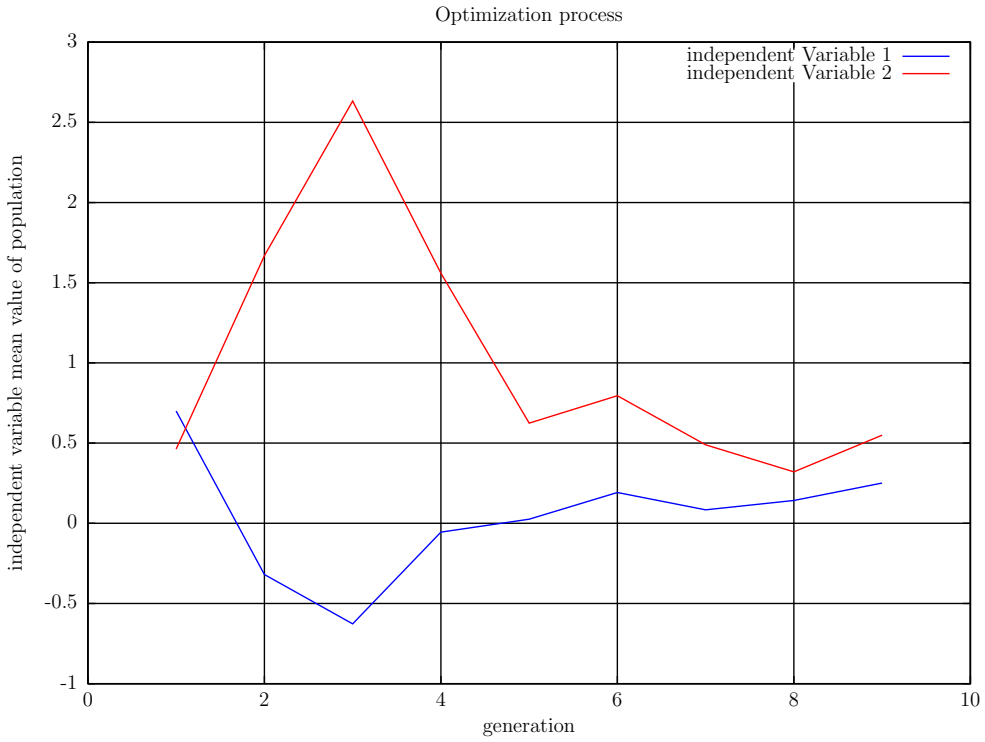


Figure 2.26 – Independent Variable Mean Value of Population function 2.50

Test function three

$$f(x) = \sum_{i=1}^{5} [x_i] \tag{2.51}$$

where $[x_i]$ represents the greatest integer less than or equal to x_i. Hence, test function three is a 5-dimensional step function. Figure 2.27 gives us an idea of how this function behaves in the problem domain.

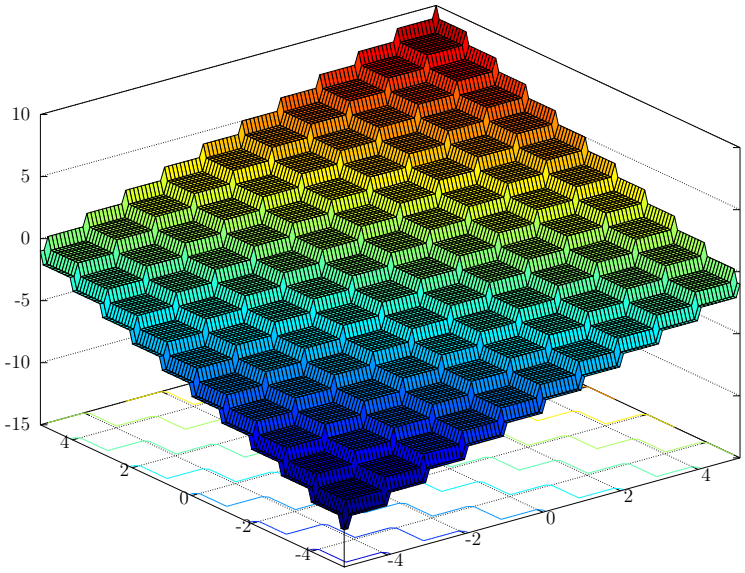


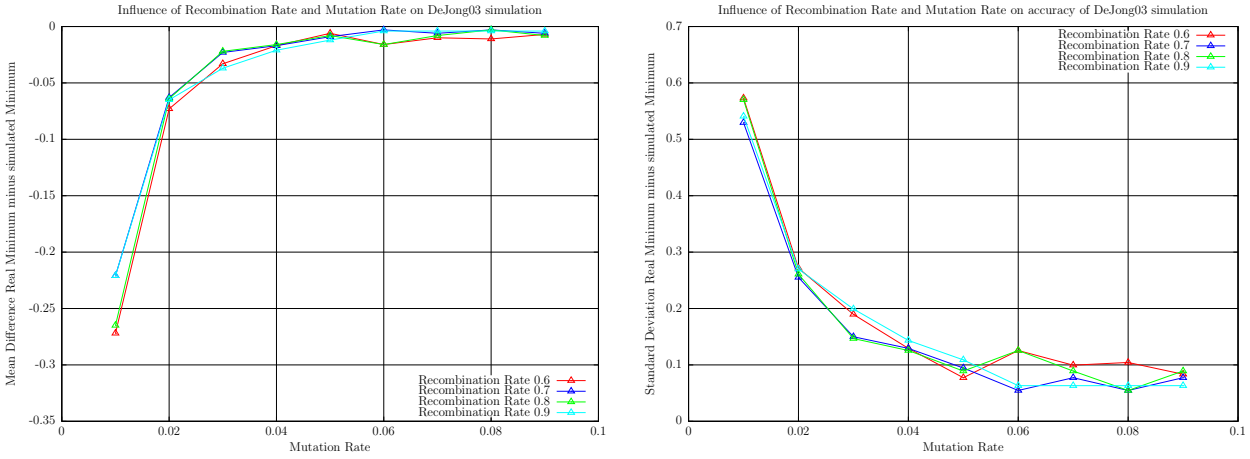
Figure 2.27 – Plot of function 2.51 in 3D - Space

Figure 2.29(a) and 2.29(b) demonstrate that a high mutation rate supports reliable results. The main issue in this problem is to prevent the algorithm to get stuck on a given level, the terraces. Because of this a high mutation rate of 0.06 is taken. The influence of the recombination rate is not that strong, to find a compromise a recombination rate of 0.7 is taken.

Table 2.14 describes the setup for a single run, as stopping criterion stability over 20 loops was taken. In figures 2.29(a) we see that the fittest population member jumps down the stairs until he's on the lowest level, figure 2.29(b) shows that this is true for the whole population. Figure 2.30 and 2.31 visualize the change in the independent variables, which at least corresponds to the dependent variable. The calculated values of the fittest chromosome were $x_1 = -5.12$, $x_2 = -5.08999$, $x_3 = -5.01495$, $x_4 = -5.07498$, $x_5 = -5.02996$, $f(x) = -30$.

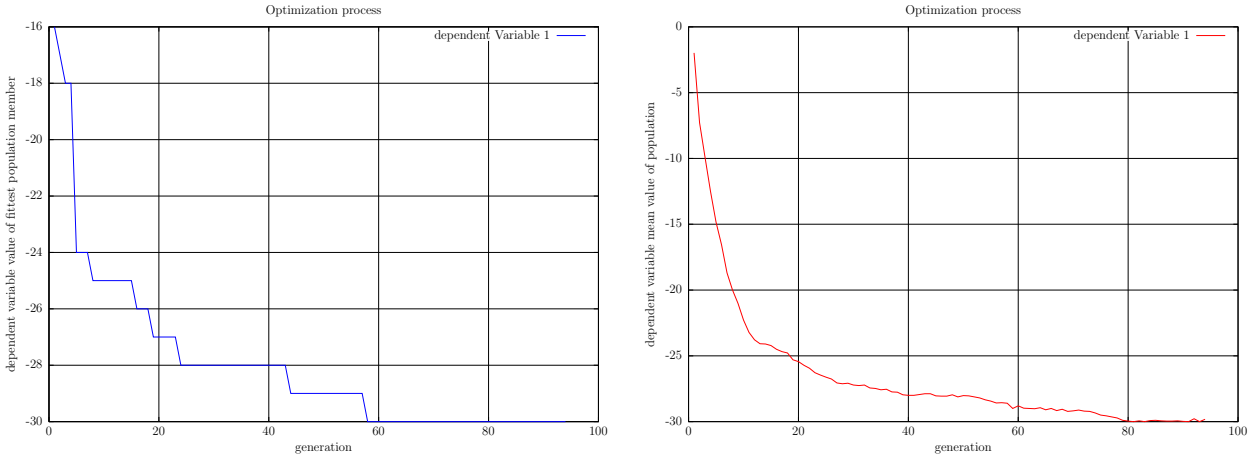
Parameter	**value**
Population size	50 chromosomes
Nbr. of generations	94
Generation gap	0.85
Mutation rate	0.06
Crossover rate	0.7
gene length	11
search space	$x_i = [-5.12 : 0.01 : 5.12]$
possible solutions	$(1024)^5$
MAX $f(5.12, 5.12, 5.12, 5.12, 5.12)$	$= 25$
MIN $f(-5.12, -5.12, -5.12, -5.12, -5.12)$	$= -30$

Table 2.13 – parameters for testing with function 2.51



(a) Mean Difference $f(x^*) - f(\hat{x})$ (b) Standard Deviation $f(x^*) - f(\hat{x})$

Figure 2.28 – Influence of Recombination and Mutation Rate on Estimation for function 2.51



(a) Fittest Population Member (b) Mean Value of Population

Figure 2.29 – Development of Dependent Variable Value in function 2.51

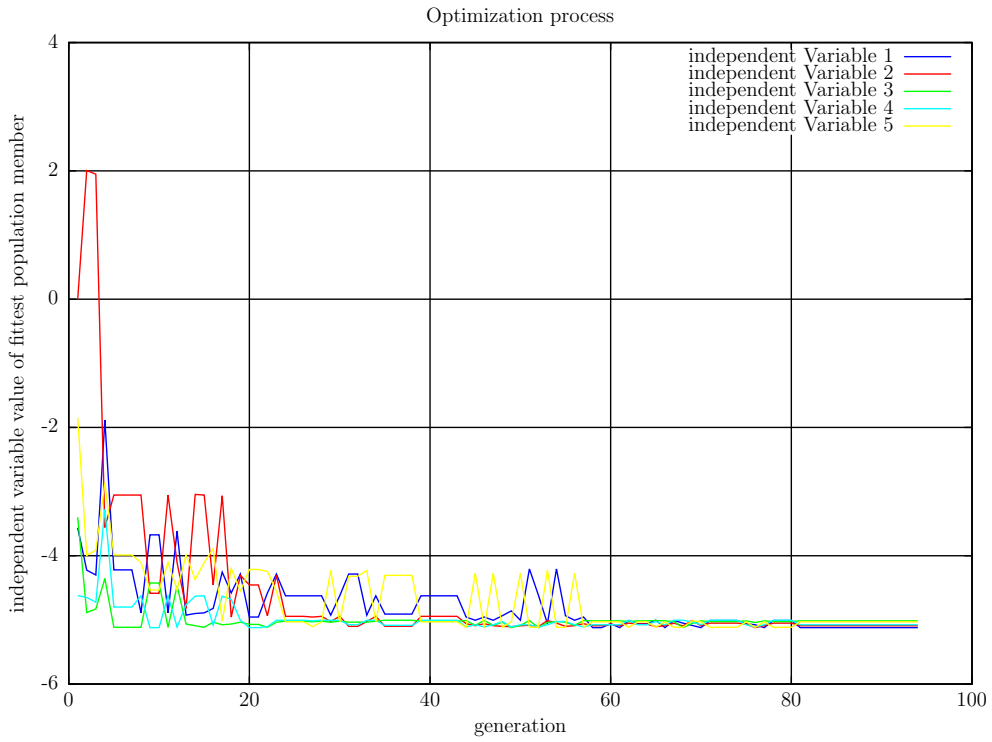


Figure 2.30 – Fittest Population Member Independent Variable Value function 2.51

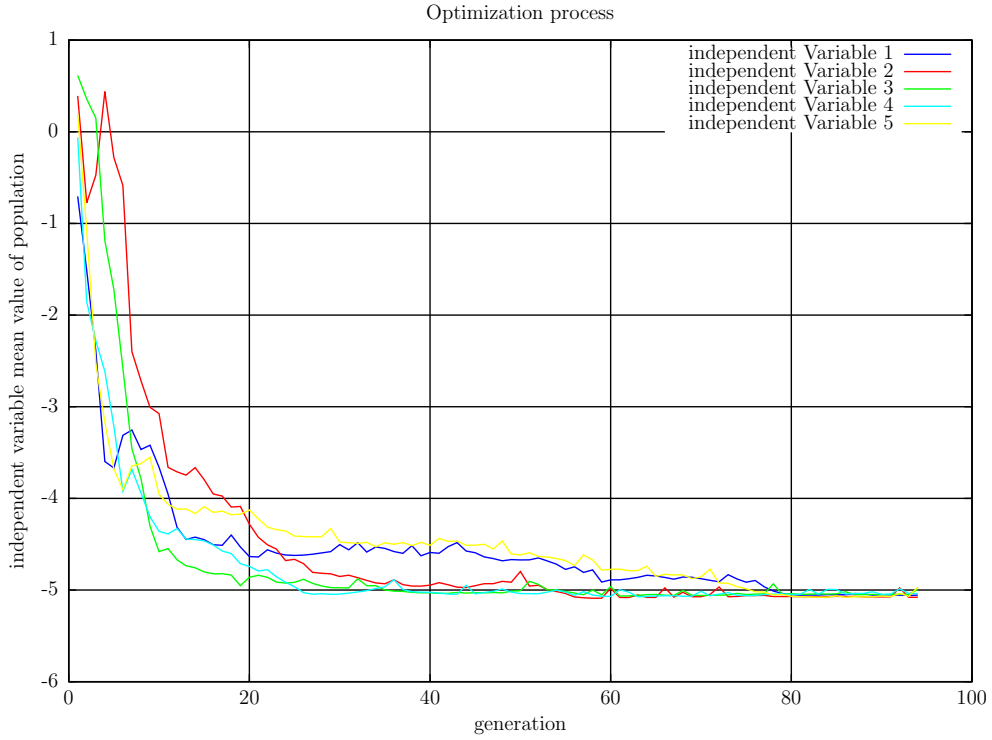


Figure 2.31 – Independent Variable Mean Value of Population function 2.51

Test function four

Test function four is given by

$$f(x) = \sum_{i=1}^{30} i * x_i^4 + GAUSS() \tag{2.52}$$

Test function four is a continuous, convex, uni modal, high-dimensional quadratic function. White Gaussian noise is added. The function as is would be a trivial task, here the noise is the main hurdle. For every generation a new set of noise was generated and added to the function values. Nevertheless the algorithm had no big problems with this task. The influence of the driving probabilities is quite strong again in this example, what can be seen from figure 2.33(a) and figure 2.33(b). The best possible combination for this problem seems to be a mutation rate of 0.04 and a crossover rate of 0.7. As in the first De Jong function, the influence of the recombination rate seems to be quite negligible. The whole population converges to the optimal solution. In

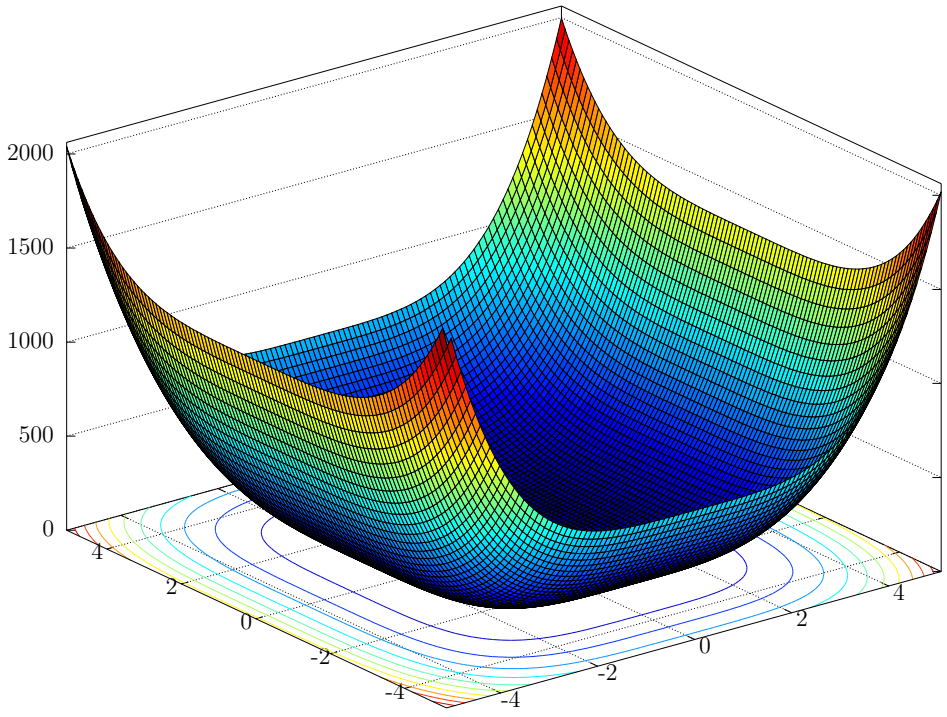


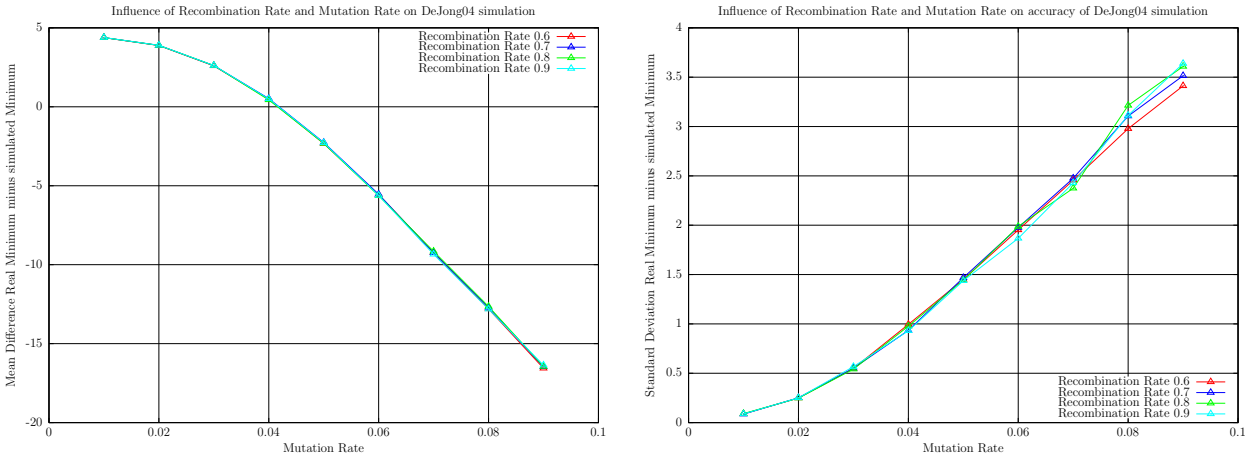
Figure 2.32 – Plot of function 2.52 in 3D - Space without Gaussion noise

contrast to the previous examples the fittest chromosome seems to be unsteady, this behavior can be explained by the noise. And still, the solution seems quite optimal. The computed value for x

was in mean -0.037, f(x) therefore was -2.781.

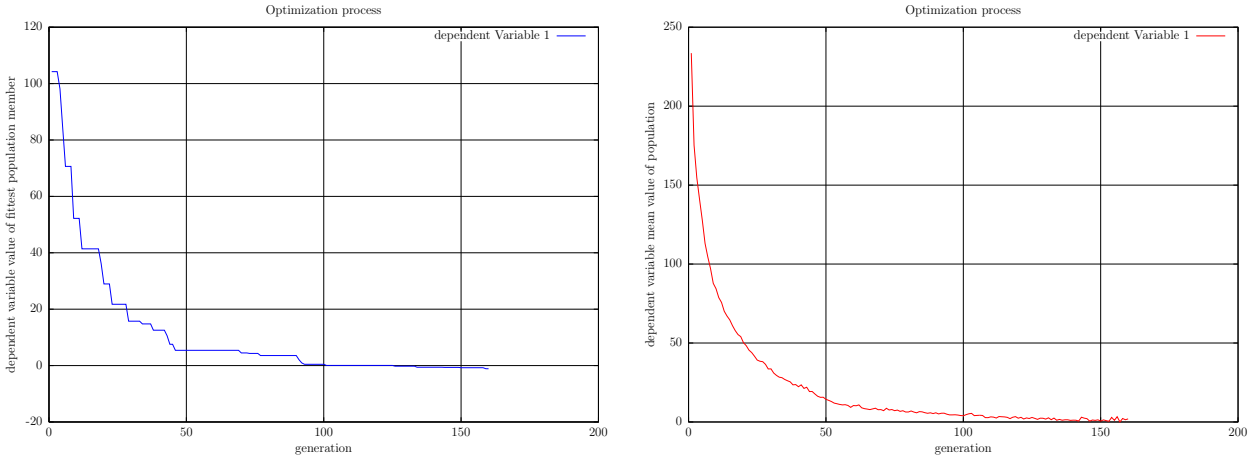
Parameter	**value**
Population size	50 chromosomes
Nbr. of generations	160
Generation gap	0.85
Mutation rate	0.04
Crossover rate	0.7
gene length	9
search space	$x_i = [-1.28 : 0.01 : 1.28]$
possible solutions	$(256)^{30}$
MAX $f(\pm 1.28, \pm 1.28, ..., \pm 1.28)$	$= 1248.2$
MIN $f(0, 0, ..., 0)$	$= 0$

Table 2.14 – parameters for testing with function 2.52



(a) Mean Difference $f(x^*) - f(\hat{x})$ (b) Standard Deviation $f(x^*) - f(\hat{x})$

Figure 2.33 – Influence of Recombination and Mutation Rate on Estimation for function 2.52



(a) Fittest Population Member (b) Mean Value of Population

Figure 2.34 – Development of Dependent Variable Value in function 2.52

Rastrigin Function

One of the most often used test functions for global optimization tasks is the Rastrigin function. In the two dimensional case it is formulated as

$$f(x) = 20 + x_1^2 + x_2^2 - 10(cos(2\pi x_1) + cos(2\pi x_2) \tag{2.53}$$

The function has many local minimas but only one global minimum at (0,0). Figure 2.35 gives an overview how the function behaves. Because of the characteristics like low dimensionality in combination with an high number of local minimums, here we implement this function. As used in a lot of publications, by using it we achive comparability to other publications. As interval for x_1 and x_2 the range $[-5.12, 5.12]$ is chosen. The function is continuous, non-convex, unimodal, low-dimensional and quadratic. No noise is added.

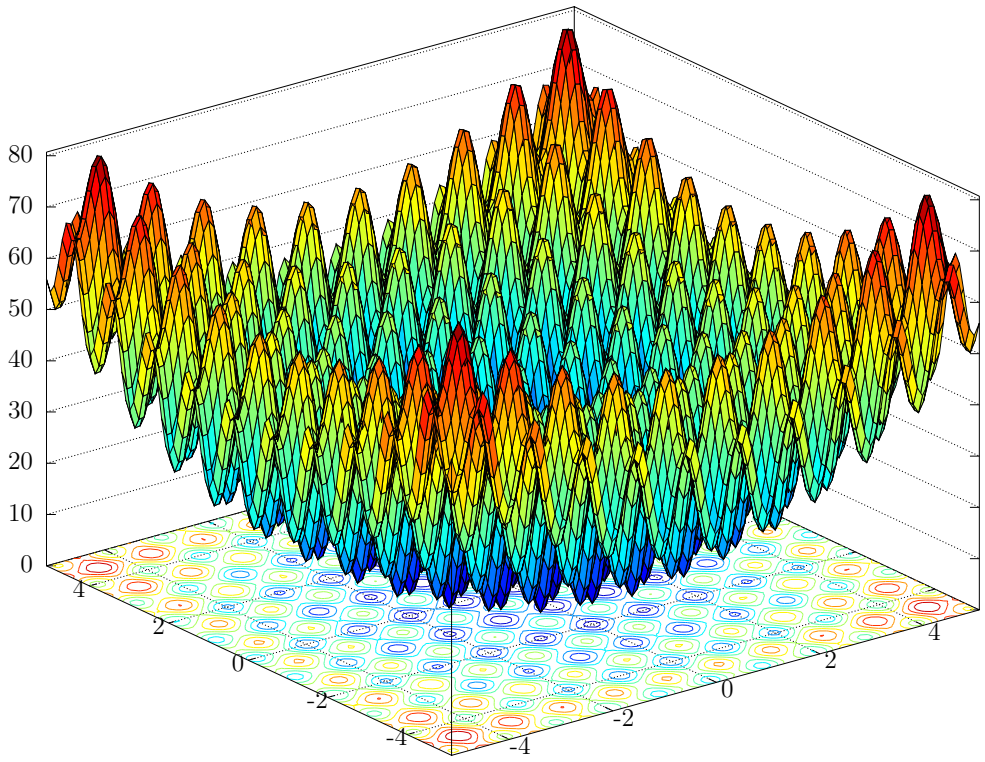
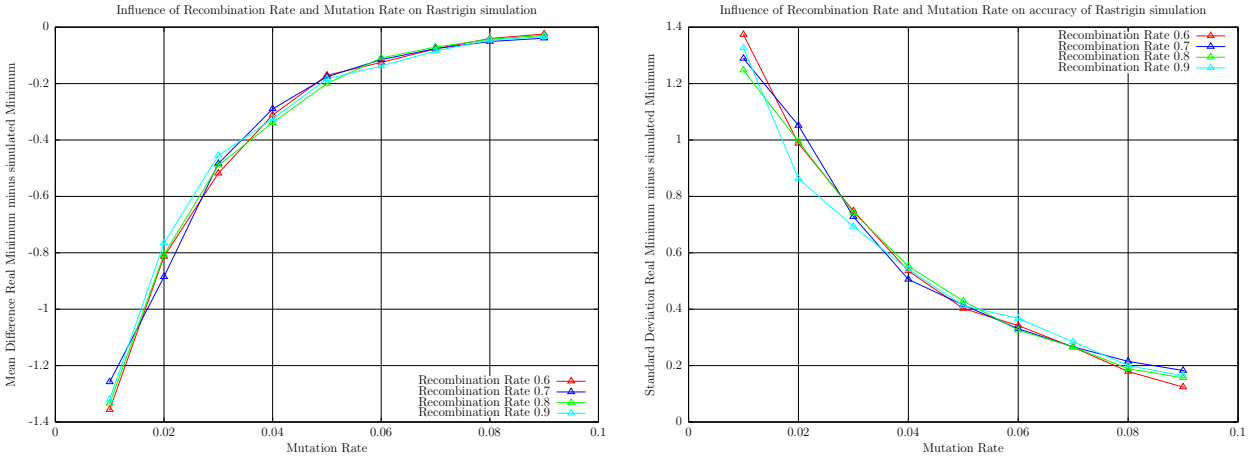


Figure 2.35 – Plot of function 2.53 in 3D - Space

Figure 2.36(a) and figure 2.36(b) show how different probabilities influence the result of the algorithm. We see clearly how the raising mutation rate has an positive influence to the evolution process. Probably the high mutation rate helps the algorithm not get stuck in one of the many local minimas. The effect of changing recombination rate in contrast seems quite low. As a result of this first test a mutation rate of 0.09 and a recombination rate of 0.6 is used for further investigation.



(a) Mean Difference $f(x^*) - f(\hat{x})$ (b) Standard Deviation $f(x^*) - f(\hat{x})$

Figure 2.36 – Influence of Recombination and Mutation Rate on Estimation for function 2.53

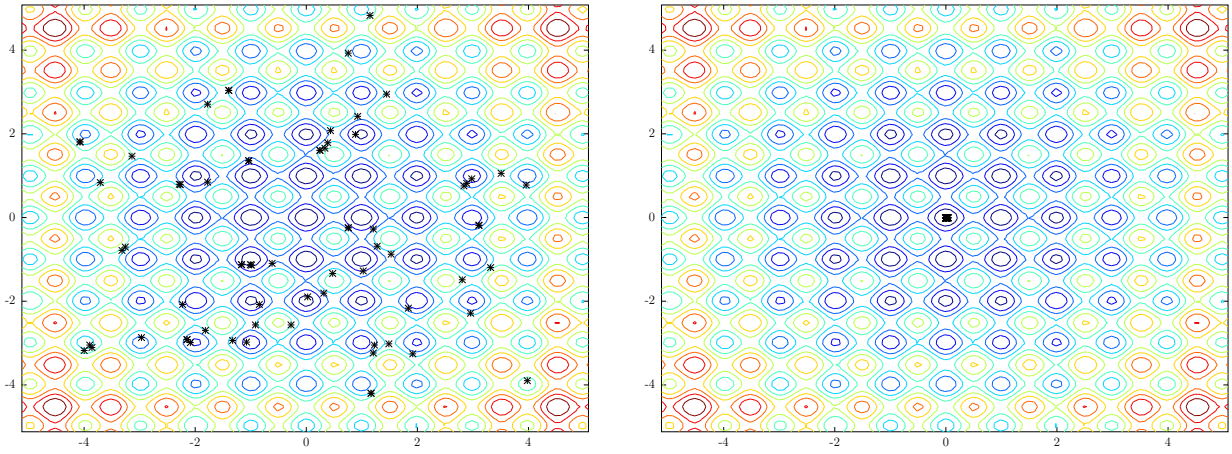
The parameters in table 2.15 were chosen to analyze an optimization process in detail. Figure 2.37(a) illustrates an initial situation, figure 2.37(b) displays a final situation (with 70 instead of 50 population members). It can be shown that a raising number of population members also has an positive effect on the quality of the result, but for staying consistent with the previous examples the population size of 50 was kept.

Parameter	**value**
Population size	50 chromosomes
Nbr. of generations	63
Generation gap	0.85
Mutation rate	0.09
Crossover rate	0.6
gene length	11
search space	$x_i = [-5.12 : 0.01 : 5.12]$
possible solutions	$(1024)^2$
MAX $f(5.12, 5.12)$	$= 25$
MIN $f(0, 0)$	$= 0$

Table 2.15 – parameters for testing with function 2.53

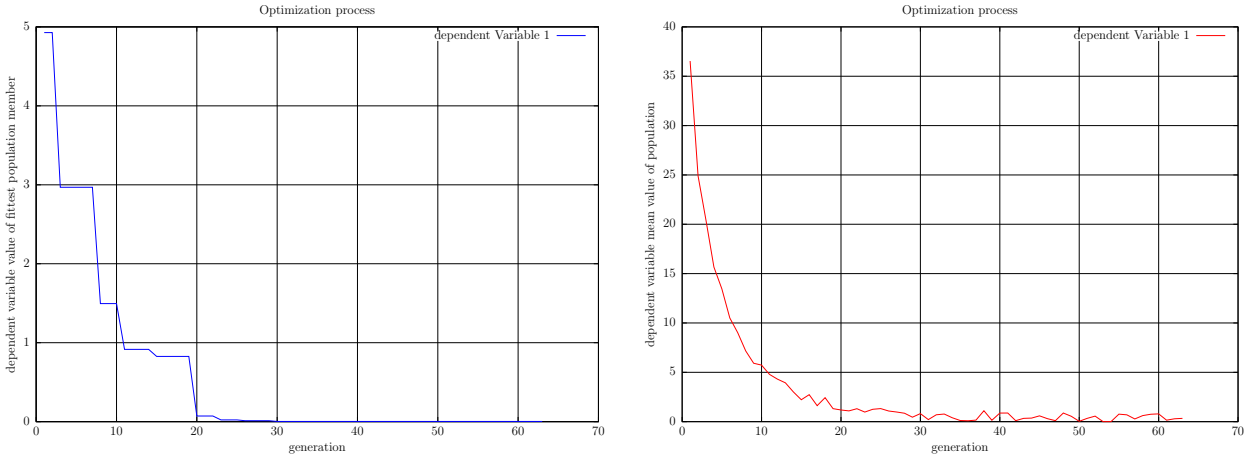
Figure 2.38(a) and figure 2.38(b) show the development of the dependent variable for such an

optimization process. Especially in figure 2.38(a) we see that the fittest population member can get stuck for some generations in a local minima. But due to the high mutation rate, in most cases the algorithm itself explores the search room and don't gets stuck. Figure 2.39 and figure 2.40 describe the independent variables for the same run. The calculated values of the fittest chromosome were $x_1 = -0.00250149$, $x_2 = -0.00250149$, $f(x) = 0.0024828$.



(a) Initial population (b) Final population

Figure 2.37 – Development of Population in search space of function 2.53



(a) Fittest Population Member (b) Mean Value of Population

Figure 2.38 – Development of Dependent Variable Value in function 2.53

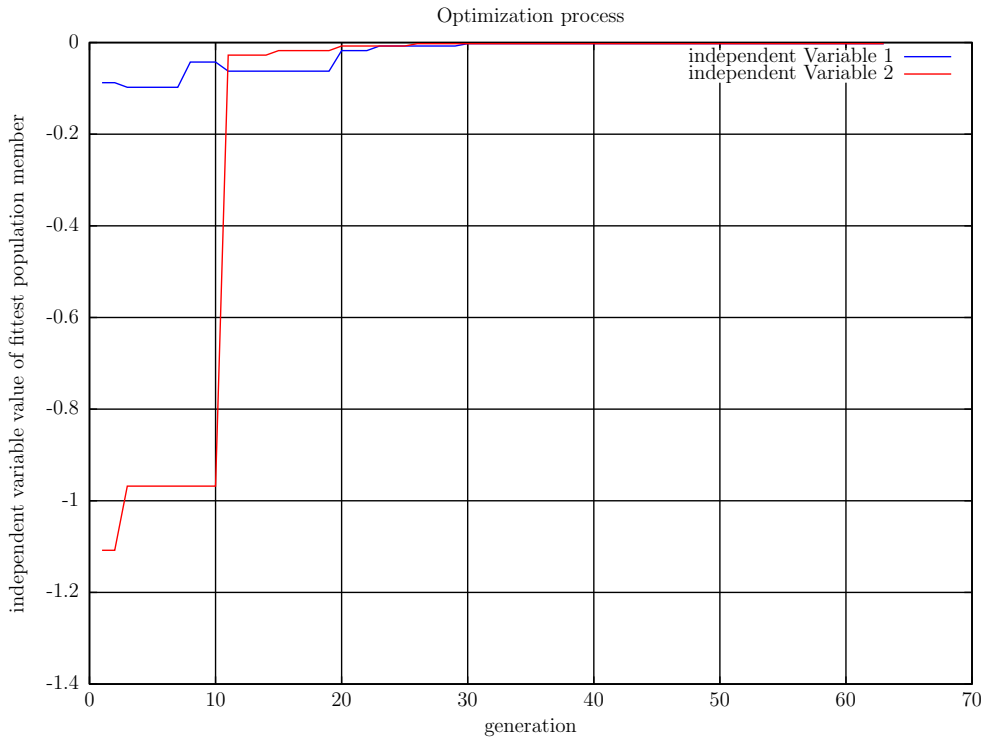


Figure 2.39 – Fittest Population Member Independent Variable Value function 2.53

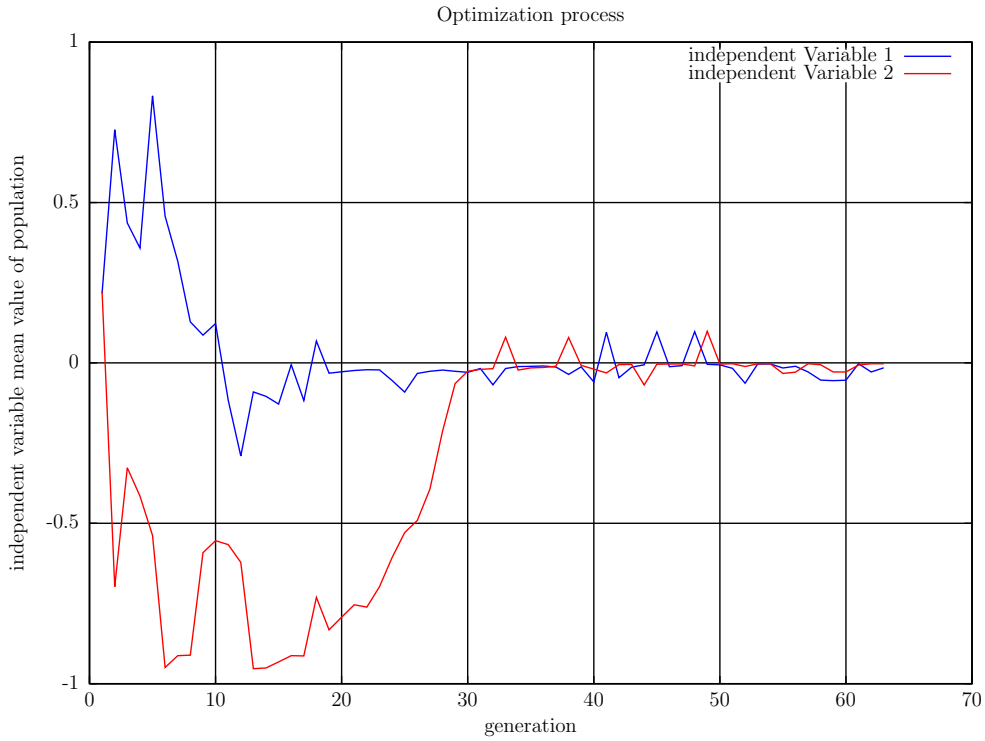


Figure 2.40 – Independent Variable Mean Value of Population function 2.53

Goldstein & Price Function

The function is given by

$$f(x_1, x_2) = [1 + (x_1 + x_2 + 1)^2 * (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$$
$$*[30 + (2x_1 - 3x_2)^2 * (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)] \quad (2.54)$$

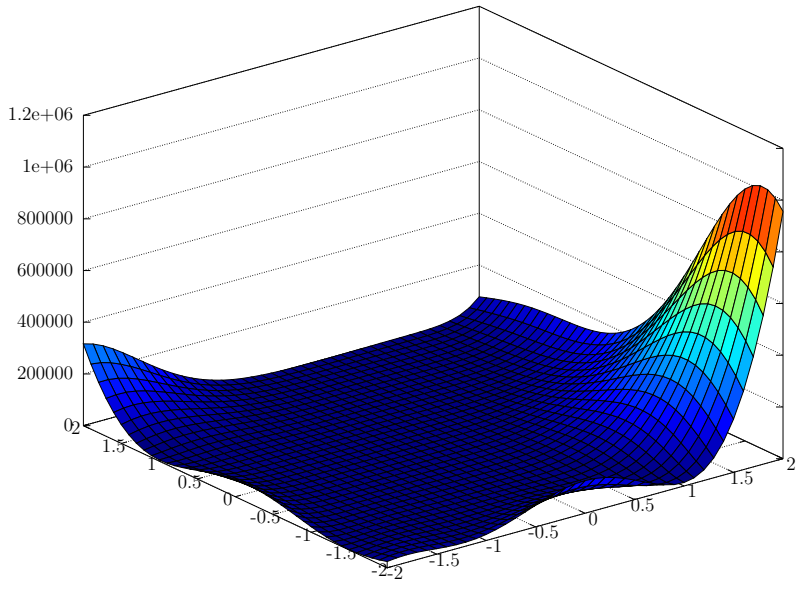


Figure 2.41 – Goldstein & Price function 2.54

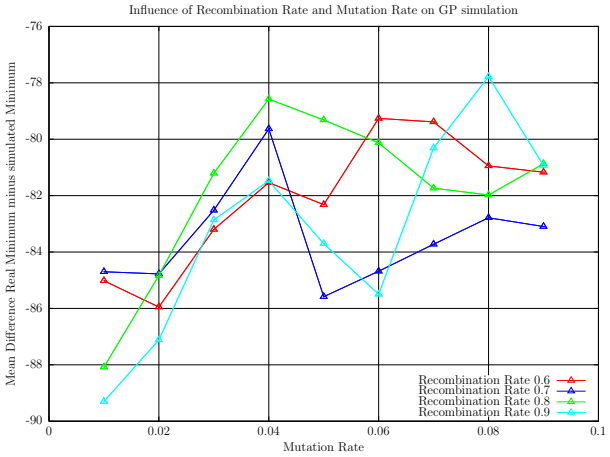
Like Rastrigin function also Goldstein & Price function is included here for giving some comparison points to other publications. Figure 2.41 gives an overview of the function behavior in space. The first impression is that most of the space is just a big valley, but this is misleading. If we take the z-axes into account we immediately recognize that the function value is steadily changing over the whole space.

The function itself seems to be a quite simple task for minimization algorithms. A first analysis with combining different probabilitis shows that for recombination rate the value of 0.9 and for mutation rate the value of 0.08 the result seems to be best fitting (compare with figure 2.42(a)and figure2.42(b)).

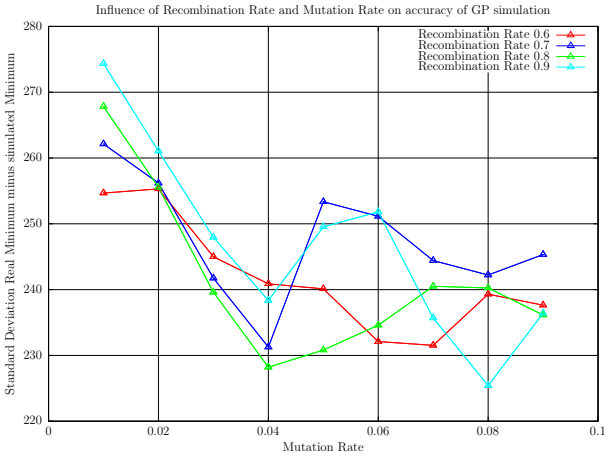
Parameter	**value**
Population size	50 chromosomes
Nbr. of generations	19
Generation gap	0.85
Mutation rate	0.08
Crossover rate	0.9
gene length	10
search space	$x_i = [-2 : 0.0078125 : 2]$
possible solutions	$(512)^2$
MIN $f(0, -1)$	$= 3$

Table 2.16 – parameters for function 2.54

Table 2.16 gives an detailed overview of the setup. In figure 2.46(a) we see that the fittest population member converges very fast to the global minimum, which is also true for the rest of the population (compare figure 2.46(b)). Figure 2.44(a) and figure 2.44(b) obtain the same trend for the independent variables.

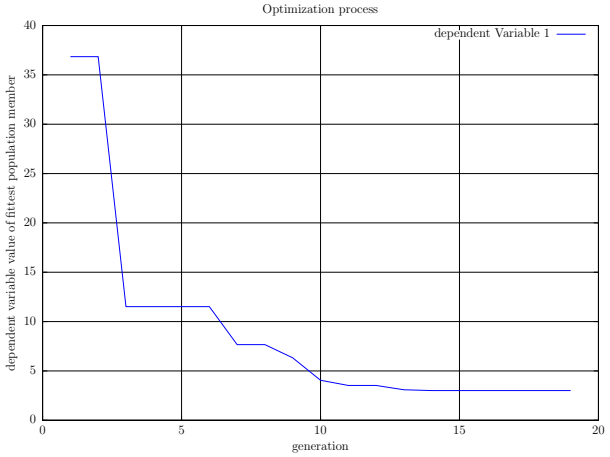


(a) Mean Difference $f(x^*) - f(\hat{x})$

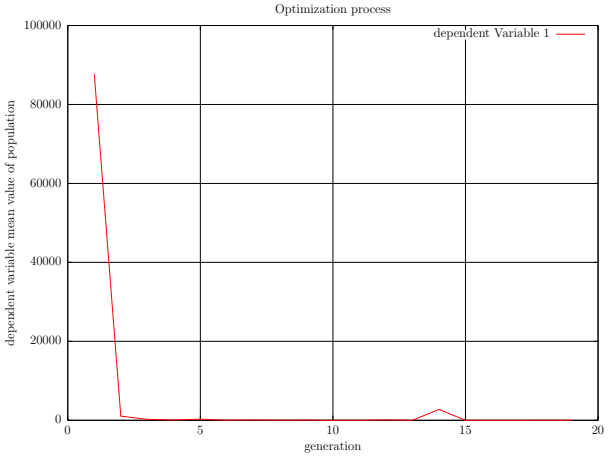


(b) Standard Deviation $f(x^*) - f(\hat{x})$

Figure 2.42 – Influence of Recombination and Mutation Rate on Estimation for function 2.54

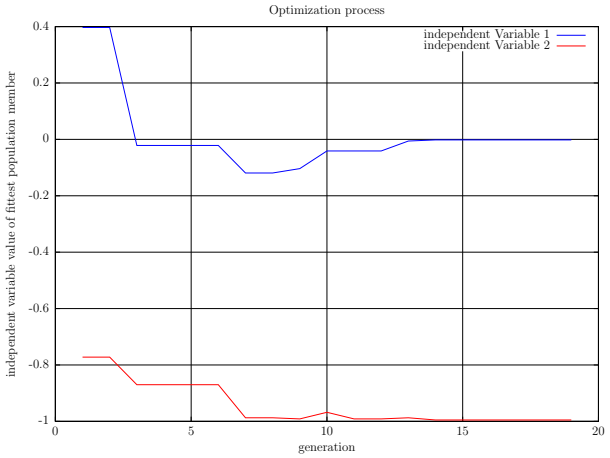


(a) Fittest Population Member

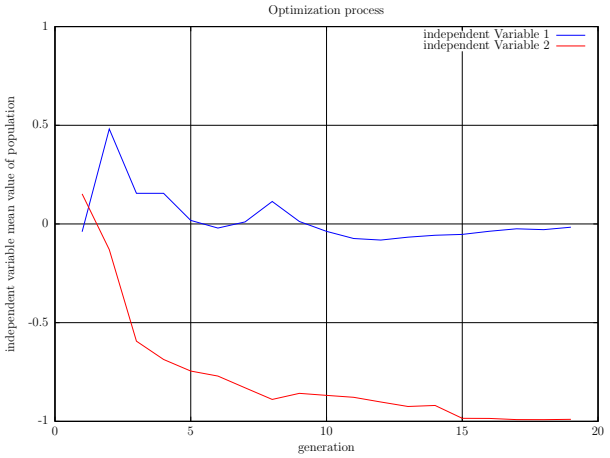


(b) Mean Value of Population

Figure 2.43 – Development of Dependent Variable Value in function 2.54



(a) Fittest Population Member



(b) Mean Value of Population

Figure 2.44 – Development of Independent Variable Value in function 2.54

Function Inversion

Function inversion is a common task in remote sensing. The previous mentioned problem domains are forward models, to use EA for solving inverse problems here an EA for solving inverse problems is stated. The goal of the algorithm is to distribute a set of points evenly on a manifold defined by $f(x, y) = c$, where $f(x, y)$ is some generating function and c is a constant [15]. As objective function

$$f(x, y) = \frac{[tanh(2 - 4x + 2y) + tanh(1 + 1x - 2y)]}{2} \tag{2.55}$$

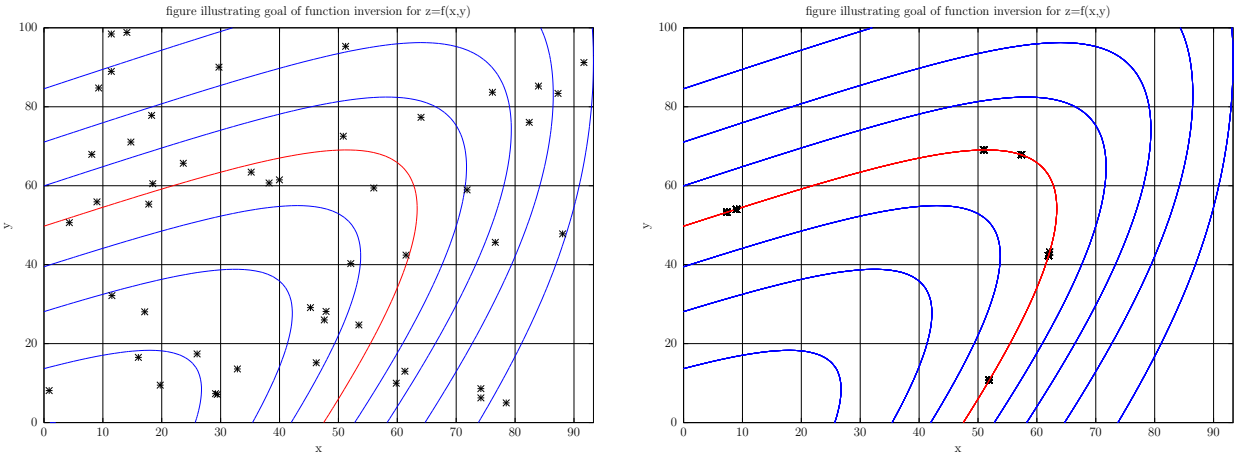
is used, the fitness function is

$$g(x) = |f(\hat{x}) - 0.5| \tag{2.56}$$

The problem is quite different to the previous ones. Here, depending on the resolution, almost infinite solutions satisfy the goal. By this example we also see how the fitness function influences the solution. Variations of the fitness functions like cosine weighting in order to get better results are possible. Reed [15] suggested to use gradient information, but for showing the basic idea behind function inversion this example is sufficient.

Parameter	**value**
Population size	50 chromosomes
Nbr. of generations	160
Generation gap	0.85
Mutation rate	0.1
Crossover rate	0.1
gene length	10
search space	$x_i = [0 : 0.002 : 1]$

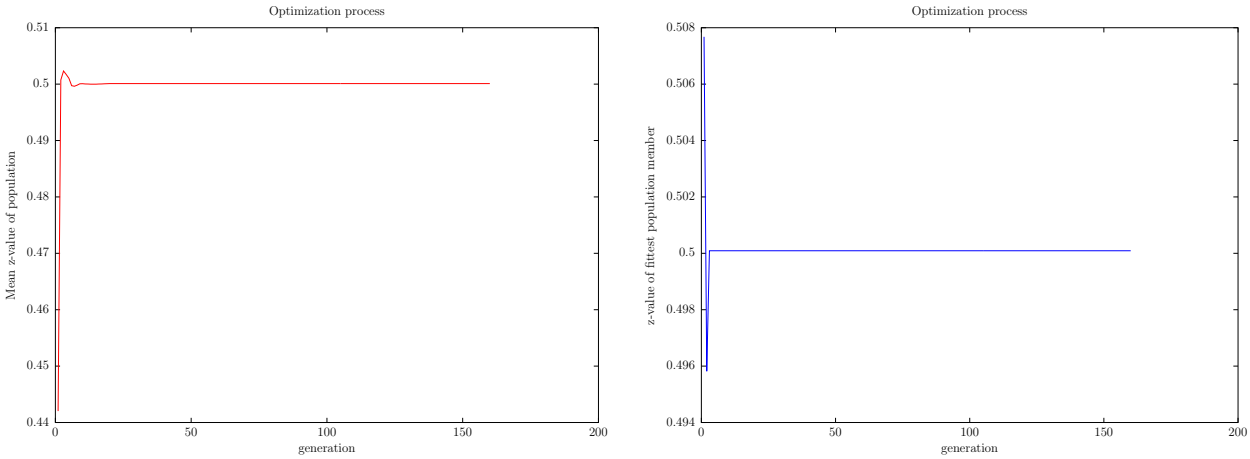
Table 2.17 – parameters for testing with function inversion example



(a) Initial distribution of population

(b) Final distribution of population

Figure 2.45 – Development of Population in search space of function 2.55



(a) Mean f(x) of population

(b) f(x) of fittest population member

Figure 2.46 – Development of Dependent Variable Value in function 2.55

Function Inversion in atmospheric Remote Sensing

As an application for the methodology of function inversion, a Remote Sensing problem is now formulated. As objective function we use the radiation around the oxygen A-band (760 nm), computed by the following parameters

- surface height (SH)
- surface albedo (SA)
- cloud top height (CTH)
- cloud geometrical thickness (CGT)
- cloud optical thickness (COT)
- solar zenith angle (SZA)
- viewing angle (VZA)
- relative azimuth angle (RAZ)

The function can then be thought of as

$$f(SH, SA, CTH, CGT, COT, SZA, VZA, RZA) = radiation[62wavelength] \tag{2.57}$$

The goal is to retrieve the three cloud parameters, which are the independent parameters in the equation . In the following examples for CTH and COT random values were substituted with lower and upper boundaries like:

The other six values were taken as known from a validation file. The file itself consists of 1000 spectra with the corresponding geophysical parameters. The algorithm was applied to all these datasets, with several combinations of recombination and mutation rates. After termination for all

Parameter	lower boundary	upper boundary
CTH	0	5
COT	0.5	1

Table 2.18 – cloud parameters and interval boundaries

of this combinations the mean difference between real parameter and estimation and the standard deviation of the estimation was computed. As fitness function an approach like in the previous example is used, just extended by taking the natural logarithm of the difference.

$$g(x) = ln|\frac{1}{f(x^*) - f(\hat{x})}| \tag{2.58}$$

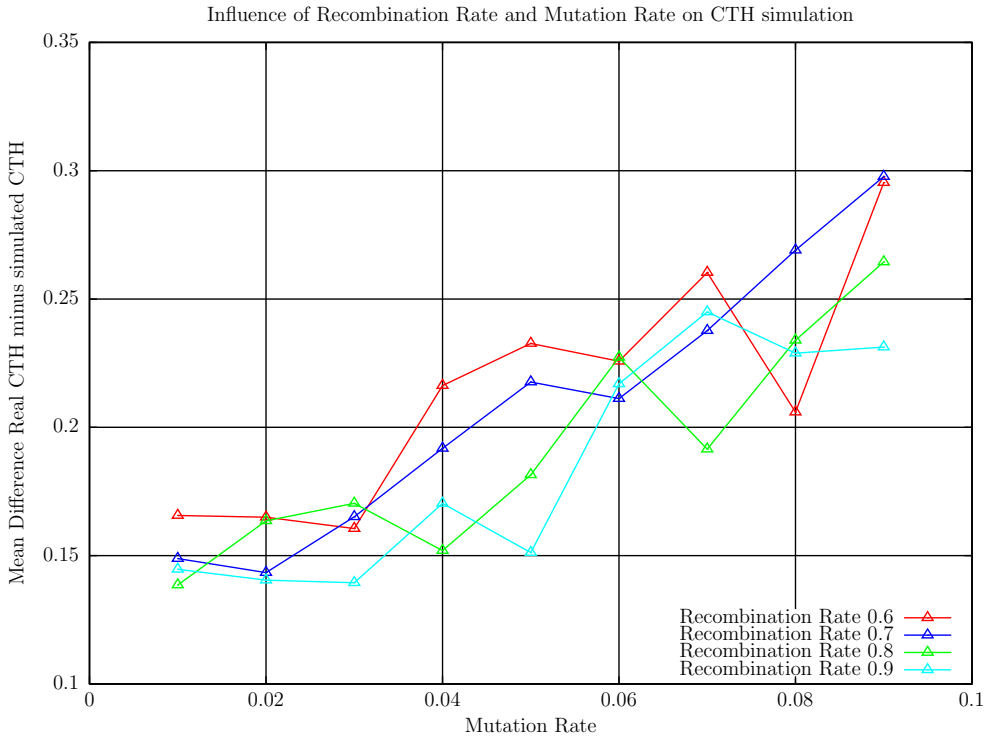


Figure 2.47 – Mean Difference CTH

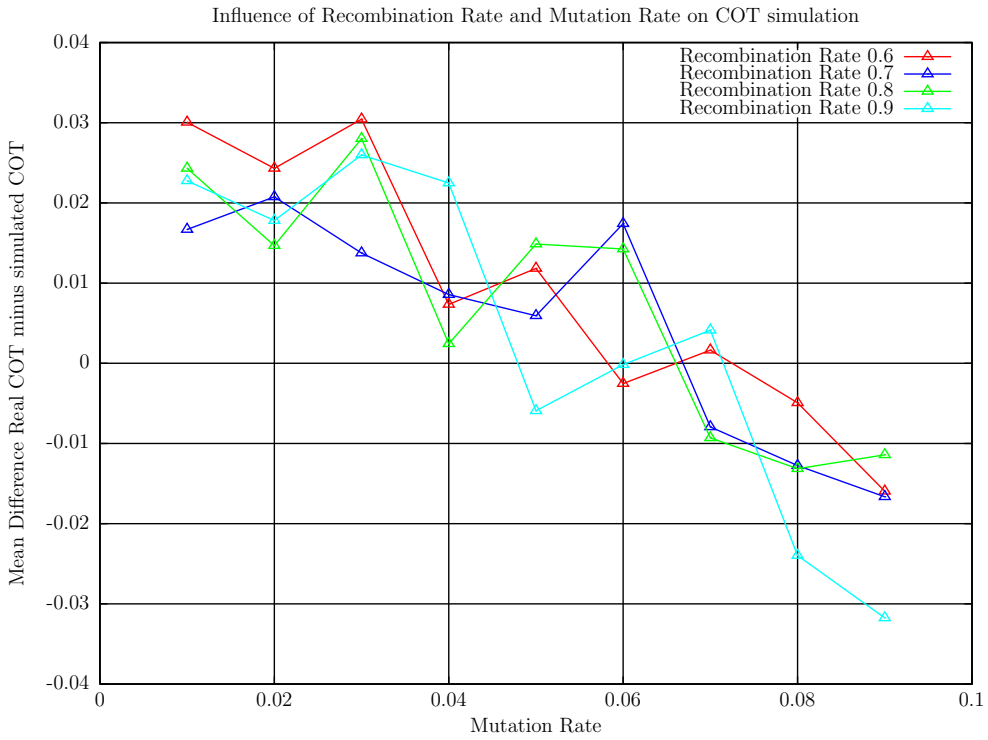


Figure 2.48 – Mean Difference COT

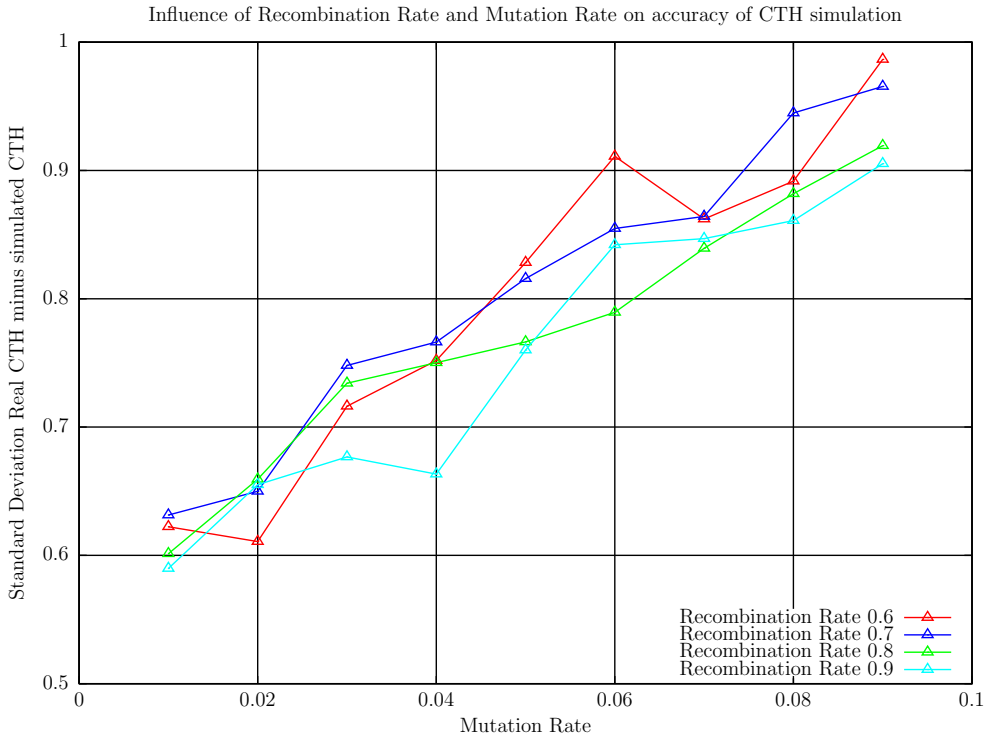


Figure 2.49 – Standard Deviation CTH f(x)

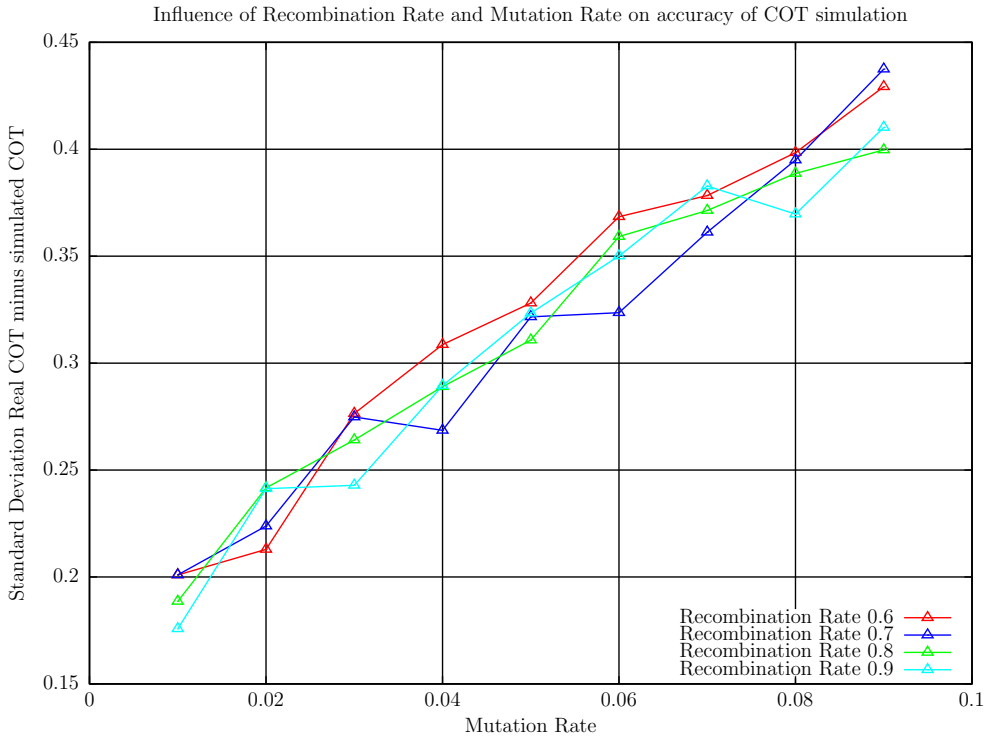


Figure 2.50 – Standard Deviation COT

2.3 Hybrid Methods

In the previous chapters deterministic and heuristic optimization methods were discussed. It was shown that for the deterministic algorithms the quality of the results and the number of iterations are related to the starting point of the algorithm and of course the problem domain itself. For the evolutionary algorithms we've illuminated the influence of the different driving probabilities and how these factors affect the quality of the results and the number of computed generations.

It is already mentioned that by applying parallelization the run time of Genetic Algorithms can be reduced significantly. In our case parallelization is implemented in the algorithm in order to make use of the fully available CPU power.

Anotherl approach for reducing the run time is the combination of deterministic and heuristic methods, so called hybrid algorithms. Both algorithm families have specific parameters which influence their behavior in the problem domain. Therefor in many cases it cannot be said which deterministic algorithm in combination with which probability values for the genetic algorithm fits best for a specific problem.

In this section a hybrid approach is introduced. The initial inspiration for this development was the GenMIN Toolbox, developed by Tsoulos and Lagaris [21]. They showed, like others [22], that by combining heuristig and deterministic search algorithms, more robust results can be achived. The genetic algorithm is enhanced with deterministic local search algorithms. The four algorithms which were included are part of GSL, they are Gradient Descend, Fletcher-Reeves, Polak-Ribiere and BFGS. The probability values for the genetic algorithm are taken from the previous results based on pure global search. As a comparison value the number of function and gradient calls is used. Besides of accuracy the present study focuses on run time reduction.

The goal of these tests is to find out, which local search algorithm fits best to which kind of problem. Therefor we let run the hybrid algorithm 1000 times for each of the four local search algorithms on the test functions. As a result, we retrieve again the mean difference between real minimum and simulated minimum and the standard deviation of real minimum minus simulated minimum. Furthermore histograms are added to show how the residuals are distributed. But besides of these plots showing the residuals of the estimation, also tables are included which show the mean number of function and derivative calls for the local search algorithm, the mean number of function calls of the genetic algorithm, the mean number of loops and mean sum of function calls for the hybrid algorithm and of course the percentage of correct estimates. As stopping criterion for the hybrid algorithm stability over seven loops was chosen.

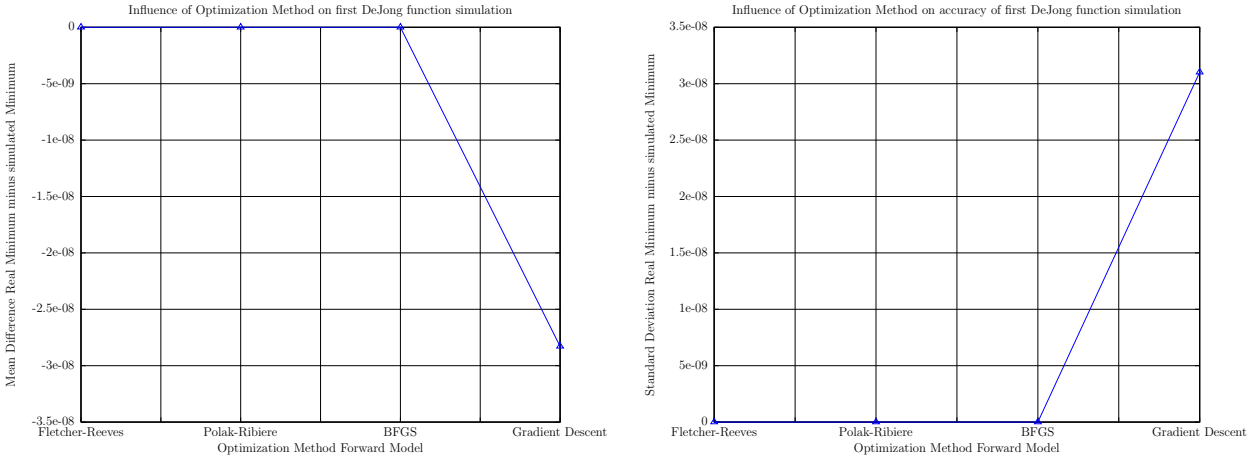
Test function one - quadratic function

The first minimization problem the hybrid algorithm is applied on is the first of the DeJong functions. This function has just one local minimum, which is also the global minimum. Because of this it's no surprise that the algorithm finds immediately the global minimum, no matter which of the determinist search algorithms is used. Table 2.19 shows the percentage of correct estimates for the four different test series.

	FR	**PR**	**BFGS**	**GD**
correct estimates (%)	100 %	100 %	100 %	100 %
mean nbr. of loops	8.72	8.7	7	7
mean f(x) calls - GA	436.1	435.25	350	350
mean f(x) calls - LS	37.4	37.46	30.1	50.75
mean f'(x) calls - LS	22.77	22.73	17.22	50.75
mean sum of calls	496.27	495.44	397.33	451.5

Table 2.19 – Analysis - Hybrid Algorithm on first DeJong function

All of the four algorithms need in mean not more than nine loops to satisfy the stopping criterion, and they find always the correct minimum. The Quasi-Newton method which uses approximations of the second derivatives needs less function calls in mean than the other three, probably because of the quadratic behaviour of the first De Jong function. Figure 2.51(a) shows the mean difference between estimation and solution, figure 2.51(b) shows the standard deviation of the estimations. Both figures confirm that the hybrid approach fits for the given task. Histogram 2.52 gives an overview of the distribution of the difference between estimations and real minimum.



(a) MeanDifference (b) Standard Deviation

Figure 2.51 – Hybrid Algorithm - local seach comparison

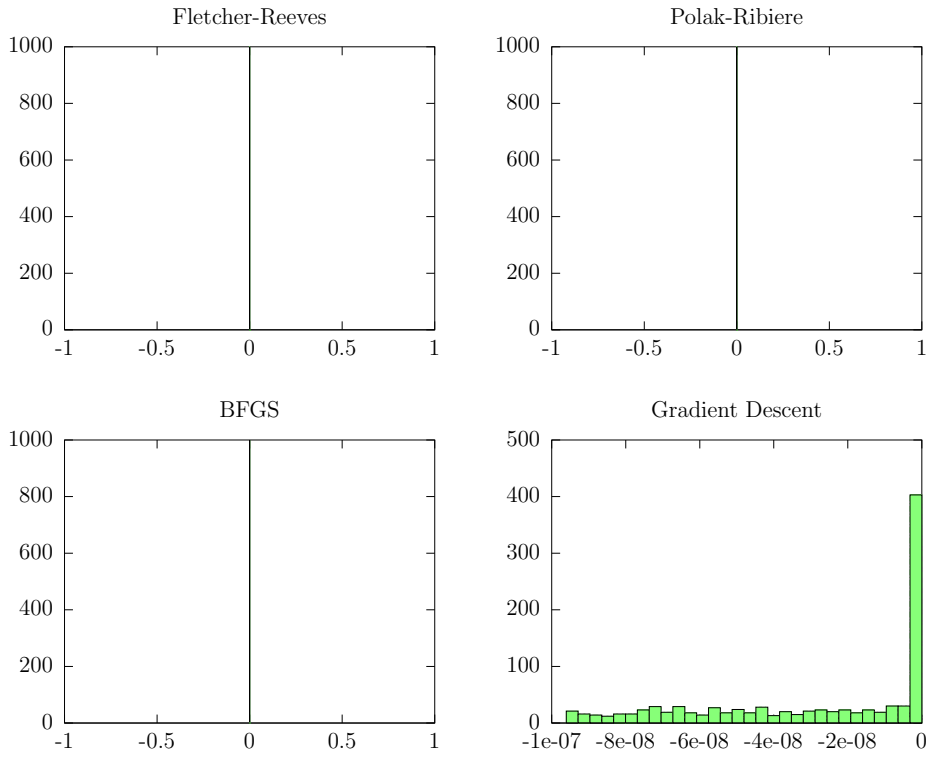


Figure 2.52 – Histogram: Real minimum - Estimated minimum

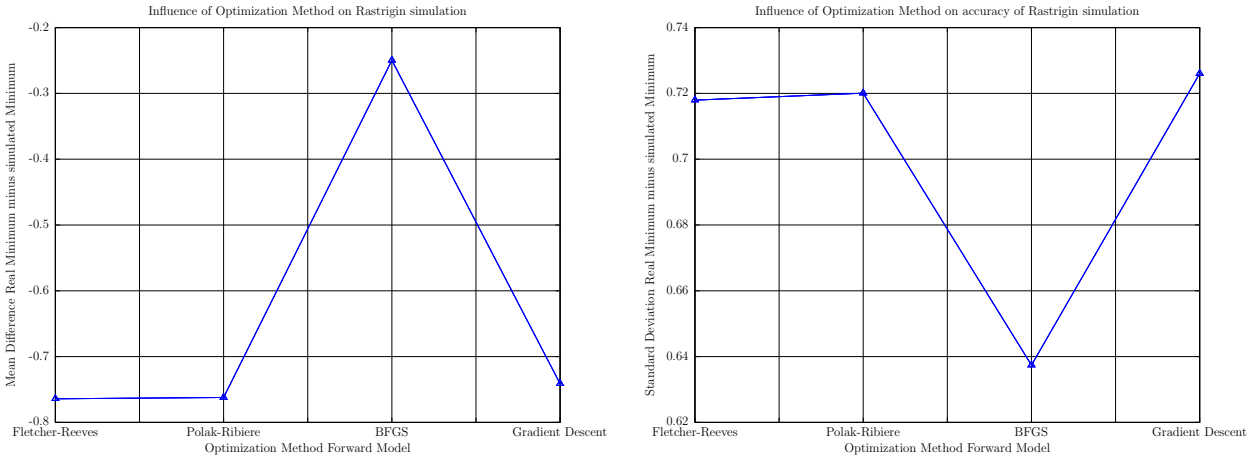
Rastrigin function

For the used two dimensional Rastrigin function, the combination of BFGS with the genetic algorithm fits best. This can be explained again by considering that the used Rastrigin function is quadratic. Because of this the used quasi Newton approach should deliver the best results. But in the test series BFGS did not just deliver better results than the other three methods, the percentage of correct estimates is more than twice as high as that by the other methods. It is worth to mentioning that BFGS in mean needs one more loop and therefore also more function calls to reach the stopping criterion, which is shown in table 2.21. But by comparing the results with the other approaches it reveals that BFGS is the only method which delivers reliable results. The other

	FR	**PR**	**BFGS**	**GD**
correct estimates (%)	37%	37.6%	86%	38.7%
mean nbr. of loops	10.52	10.38	11.53	10.31
mean f(x) calls - GA	526	518.8	576.8	515.65
mean f(x) calls - LS	235.34	232.98	414.44	121.82
mean f'(x) calls - LS	53.45	52.81	62.17	121.82
mean sum of function calls	814.78	804.59	1053.4	759.29

Table 2.20 – Analysis - Hybrid Algorithm on Rastrigin function

methods differ just slightly in the number of function calls and quality of estimation. The histograms in figure 2.54 which shows the distribution of the differences between real minimum and estimated minimum strongen the conviction that BFGS is the best choice in quadratic problem domains. In contrast to the gradient and conjugate gradient methods no outlier is more distant than -2 from the real minimum. Figure 2.53(a) and figure 2.53(b) complete the results gained by this test series.



(a) MeanDifference (b) Standard Deviation

Figure 2.53 – Hybrid Algorithm - local seach comparison

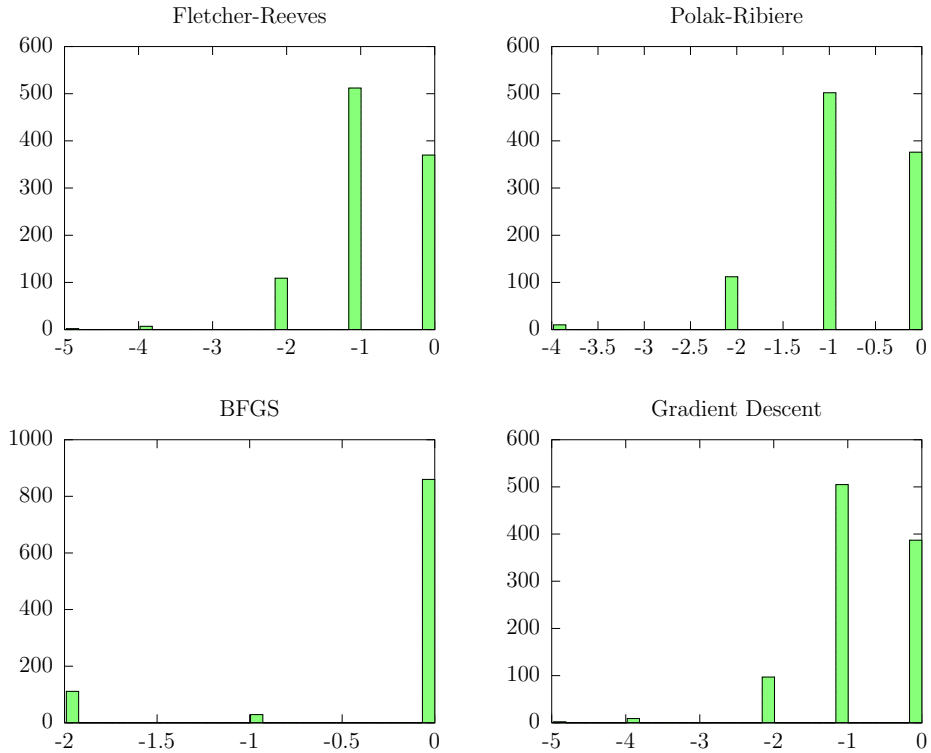


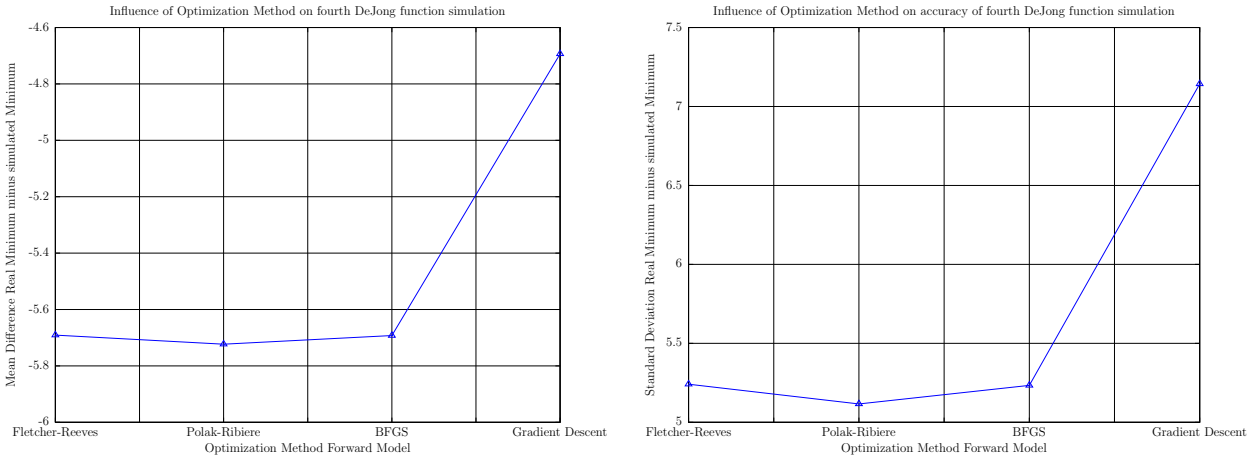
Figure 2.54 – Histogram: Real minimum - Estimated minimum

Test function four - function with noise

The hybrid algorithms in general had problems with the fourth De Jong function. The weakness of the algorithm can be explained by considering that this function has 30 dimensions, moreover to each dimension gaussion noise is added. As table 2.21 shows, the different approaches do not differ to much in the number of function calls and percentage of correct estimates. But by analyzing figure 2.55(a), figure 2.55(b) and figure 2.56 the conclusion can be made that Gradient Descent method in this case fits better than the other three more complex methods. The Standard deviation is highest with Gradient Descent, but the mean difference is smallest which is also shown by the histograms of the residuals.

	FR	**PR**	**BFGS**	**GD**
correct estimates (%)	1.6%	0.4%	1.4%	1.1%
mean nbr. of loops	12.55	12.66	12.57	14.56
mean f(x) calls - GA	627.6	633.2	628.55	727.9
mean f(x) calls - LS	461.94	463.48	458.91	215.96
mean f'(x) calls - LS	38.97	37.55	33.91	215.96
mean sum of function calls	1128.5	1134.2	1121.4	1159.8

Table 2.21 – Analysis - Hybrid Algorithm on fourth De Jong function



(a) MeanDifference (b) Standard Deviation

Figure 2.55 – Hybrid Algorithm - local seach comparison

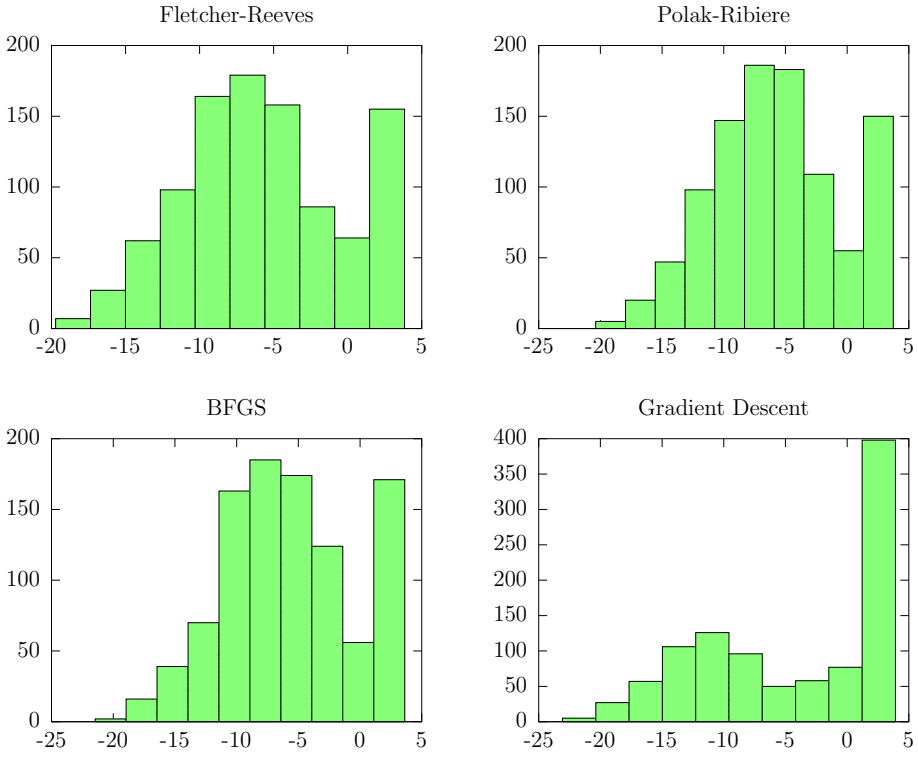


Figure 2.56 – Histogram: Real minimum - Estimated minimum

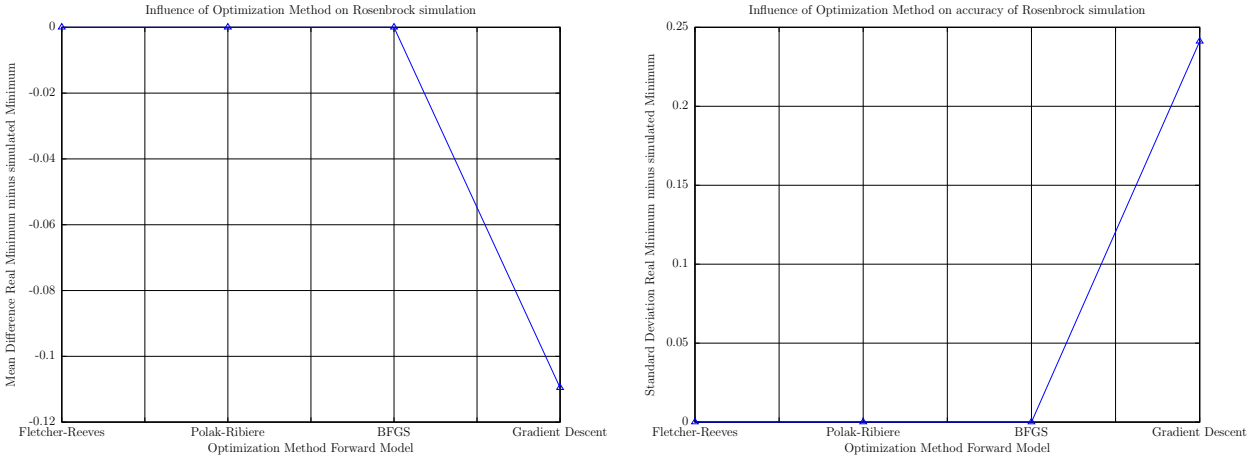
Rosenbrock function

Applying the hybrid algorithms to Rosenbrock function it delivered no surprises. Gradient Descent method couldn't compete in this problem with the other three algorithms. The reason gets clear by watching figure 2.22. In the long banana valley the algorithm does not gains enough gradient information. This lack of information is not balanced by the genetic algorithm, because of this the Gradient Descent Solution needs in mean more function calls for less accurate solutions. Details are shown in table 2.22.

	FR	**PR**	**BFGS**	**GD**
correct estimates (%)	100 %	100 %	100 %	14.6 %
mean nbr. of loops	7.11	7.07	7.08	10.99
mean f(x) calls - GA	355.7	353.5	354.3	549.6
mean f(x) calls - LS	159.66	154.99	186.34	1185.2
mean f'(x) calls - LS	131.53	127.56	125.81	1185.2
mean sum of function calls	646.89	636.05	666.45	2920

Table 2.22 – Analysis - Hybrid Algorithm on Rosenbrock function

Figures 2.57(a), 2.57(b) and 2.58 support the conclusion that despite of Gradient Descent all other three methods are able to solve this minimization problem.



(a) MeanDifference (b) Standard Deviation

Figure 2.57 – Hybrid Algorithm - local seach comparison

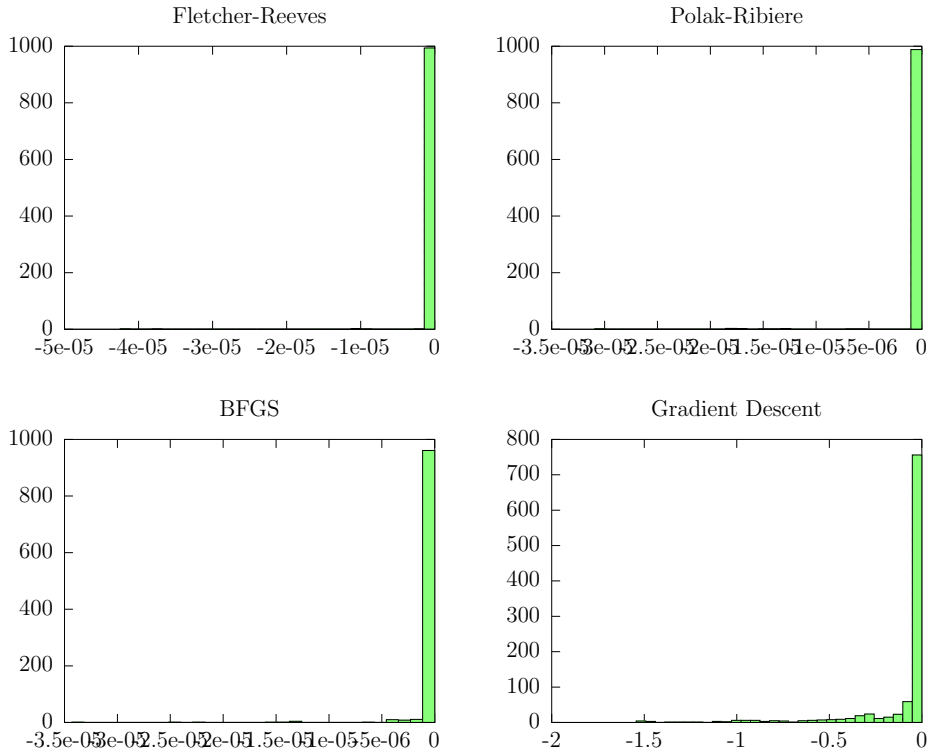


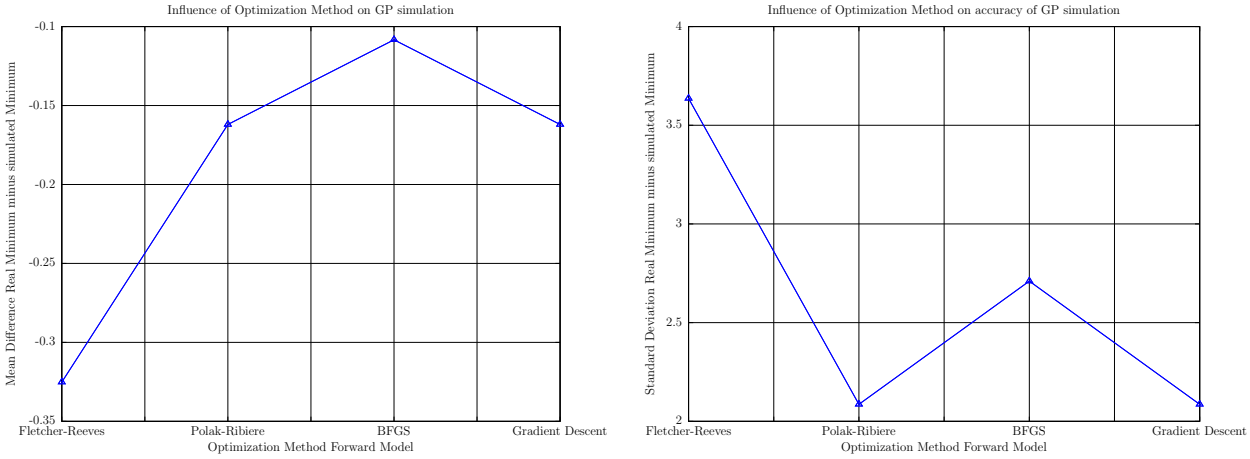
Figure 2.58 – Histogram: Real minimum - Estimated minimum

Goldstein & Price Function

In contrast to Rosenbrock function in solving the Goldstein & Price Function the Gradient Descent method delivered competitive results. Like BFGS in mean it took just seven loops for finding the global minimum. Surprisingly this was the only test function despite the first De Jong function were all four hybrid algorithms succeeded.

	FR	**PR**	**BFGS**	**GD**
correct estimates (%)	100 %	100 %	100 %	100 %
mean nbr. of loops	8.72	8.7	7	7
mean f(x) calls - GA	436.1	435.25	350	350
mean f(x) calls - LS	37.4	37.46	30.1	50.75
mean f'(x) calls - LS	22.77	22.73	17.22	50.75
mean sum of function calls	496.27	495.44	397.33	451.5

Table 2.23 – Analysis - Hybrid Algorithm on Goldstein & Price function



(a) MeanDifference (b) Standard Deviation

Figure 2.59 – Hybrid Algorithm - local seach comparison

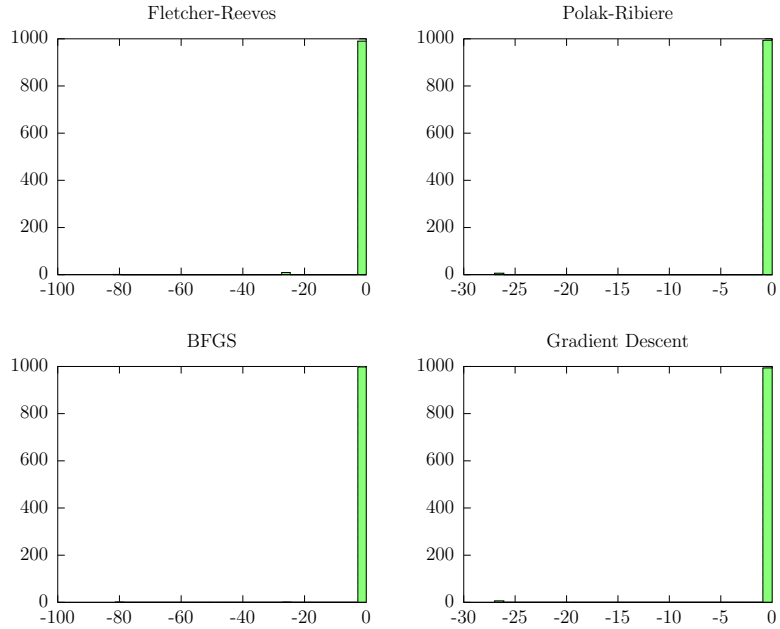


Figure 2.60 – Histogram: Real minimum - Estimated minimum

2.4 Conclusions concerning the methodology

Some conclusions can be reached already after this methodology part.

First, deterministic algorithms for multidimensional minimization in the form of $\min_{x\in\mathbb{R}^n} f(x)$ were applied on a set of test functions. These algorithms differ in computing the search direction and step length. Because of this it can not be guaranteed that equal results are gained, nor that they need an equal number of iterations until termination. Furthermore the influence of the starting point in local search was shown.

Second, the evolutionary computing framework and the operators of genetic programming were introduced. It's impossible to give general advices on how to set the probabilities for the genetic algorithm. It has been seen that results are strongly problem dependent. Nevertheless it seems that higher mutation rates fit better for problems with lots of local minimas. The genetic algorithm is quite fast in finding high quality regions in search space for optimization. It was shown, that by reformulating the fitness function not just problems in the form $\min_{x\in\mathbb{R}^n} f(x)$ but also $\min_{x\in\mathbb{R}^n} |y - f(\hat{x})|^2$ can be solved.

Parallelization on thread level was added to the genetic algorithm. It was shown that GA are in general easy to parallelize. The object oriented programming style was beneficial in this task. The decrease in runtime is, depending on the computer, remarkable.

Hybridization of the genetic algorithm was done with the different multidimensional minimizers of GSL. The comparison of the different local search algorithms gives clear results. Here by comparison we see that BFGS is superiour to the other three methods not just by quality of the estimate, but also by means requiring less function calls than the competitors. Probably because of this in comparable hybrid algorithms like GenMIN [21] just BFGS is used for local search.

The genetic algorithm is quite fast in finding the most promising regions for global optimization in the search space. In these regions the deterministic algorithms converge much faster to the minimum [22]. This leads to the conclusion that in general a hybrid algorithm delivers more reliable results than pure genetic algorithms, and this results are reached by needing less function calls.

3 Optimization of an ozone retrieval algorithm

In this part the optimization is a combinatorical task. We have level one data from GOME[1], which are processed to level 2 data using UPAS. The UPAS system gets as input not just the filename of the orbit, which it should process, furthermore as argument a vector containing 88 numbers is passed. The numbers are just zeros and ones. For the processing with UPAS 88 spectral channels in the range from 325 nm to 335 nm are of interest, the vector indicates with an one that a channel should be used, a zero indicates that a channel shouldn't be used. The level two product is stored as a HDF[2] file, the interesting part of this product is the total ozone column in dobson units, which is stored in an vector.

The best accuracy can be achived by using each of the available spectral bands. By using less spectral bands the computational effort can be minimized, but of course also for the loss of accuracy. The question here is, if we take a fixed number of spectral channels, for example 40, which combination of the channels gives the most accurate result? For solving these task analytically, each possible combination with 40 ones and 48 zeros has to be computed. Then the residuals between the computed total ozone columns and the resulting total ozone column for using all measurements have to be computed. The level two dataset with the smallest residuals belongs then to the optimal combination of zeros and ones in the input vector. The number of possible combinations is given by the binomial coefficient

$$C_{88}^{40} = \frac{88!}{40!(88-40)!} = 1.83E + 25 \tag{3.1}$$

The processing from level one to level two with UPAS with a given input vector takes about 20 minutes. This means, that computing each combination would take $6.96E + 20$ years. Because of this it's impossible to calculate each combination within an adequate time, even if we work on multi-core platforms.

[1] Global Ozone Monitoring Experiment
[2] Hierarchical Data Format

3.1 Algorithm design

In contrast to calculating each possible combination and its corresponding total ozone column values, an evolution strategy may solve the search for the best input vector quite fast. The idea is to have a population, each chromosome of this population represents an input vector. The independent variable is the input vector, the dependent variables are the retrieved ozone column values. Figure 3.3(b) shows an random input vector, the bars show the positions of the ones in the vector. Figure 3.3(a) shows the input vector for computing the reference ozone column, for which all 88 measurements were used. The fitness of each chromosome is calculated by taking the sum of the squared residuals between computed total ozone column and the reference total ozone column.

$$r = \sum_{i=1}^{1428} |(y_i - f(x_i))^2| \tag{3.2}$$

This value, and also the residual for each single ozone measurement, is also part of each chromosome. In each generation the evolutionary operators selection, recombination and mutation are applied, with the restriction that the number of ones in the input vector stays equal. The objective function which is most time consuming is called in parallel. Therefore the not yet in Boost included Boost.Process library[3] is used. This library offers the ability to call executables, pass arguments to these executables and to wait for the running processes until they finish. In this case we called UPAS, passed for each chromosome the included input vector and then waited for the results. The working processes were started in parallel and taken out on 4 blades of a blade server, each having 12 cores. Because of this a population size of 48 was chosen. After termination of the working threads the results of the single working processes were written in separate directories in hdf - files. These files were then read and the retrieved ozone values were used for updating the values in the O3_Total_Column vector. For reading these hdf-files, the freely available hdf-library[4] was used. Figure 3.1 gives an overview of the used classes and their member variables and methods.

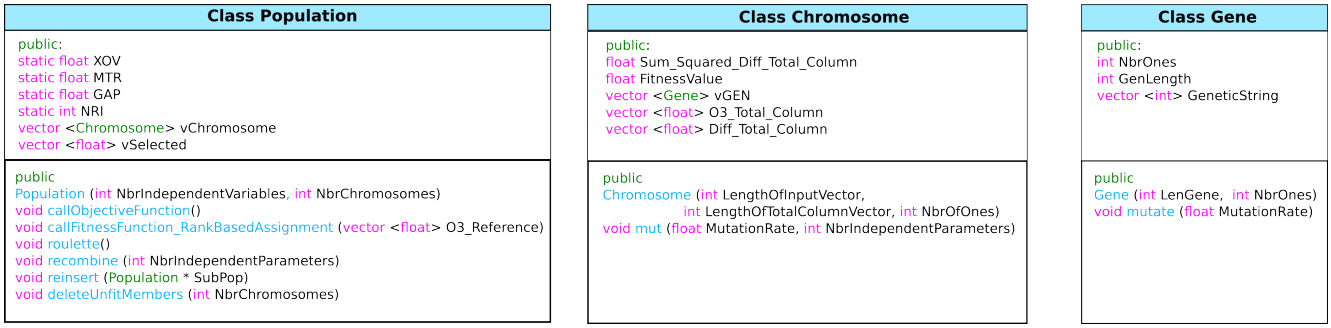
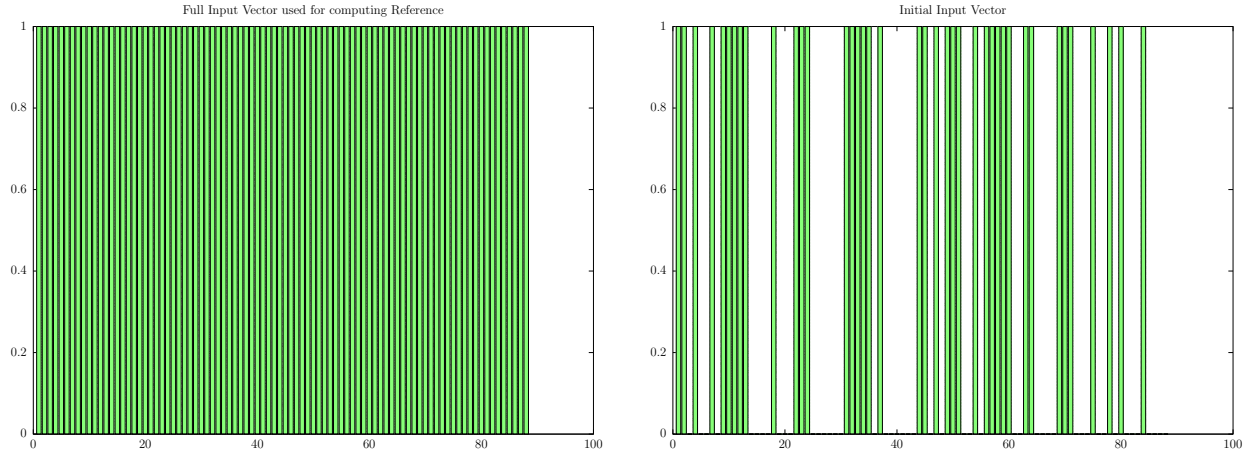


Figure 3.1 – Class diagram O3

The stopping criterion is that the residuals of the fittest population member do not decrease

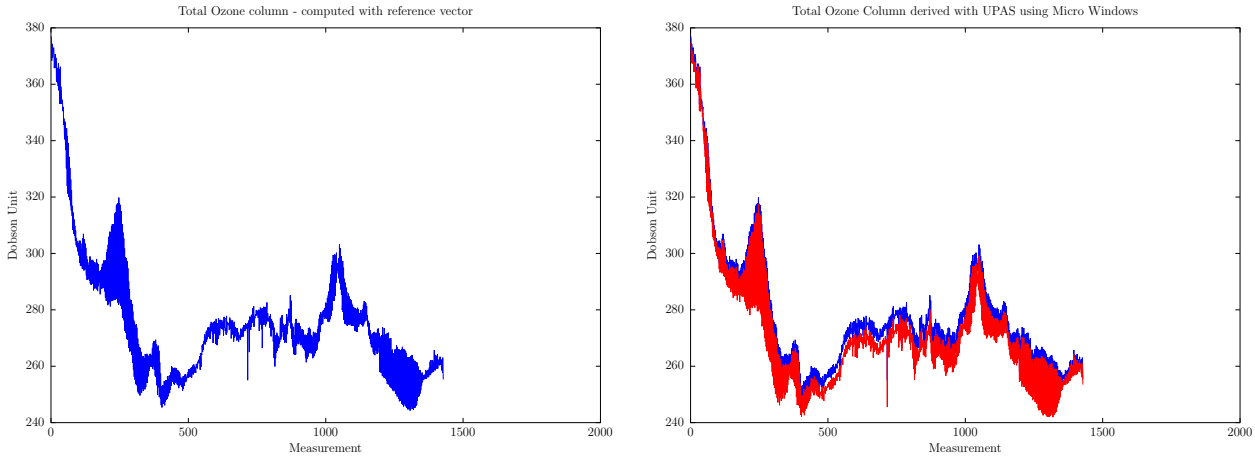
[3]available from `http://www.highscore.de/boost/process0.5/` - 23.05.2013
[4]available from `http://www.hdfgroup.org/` - 23.05.2013

over seven loops. Figure 3.3(a) shows the reference ozone values, figure 3.3(b) shows the reference spectra and the computed spectra, which belongs to the input vector of figure 3.3(b). The residuals in this case are 21109.



(a) Reference Input Vector (b) Random Input Vector

Figure 3.2 – Input Vectors



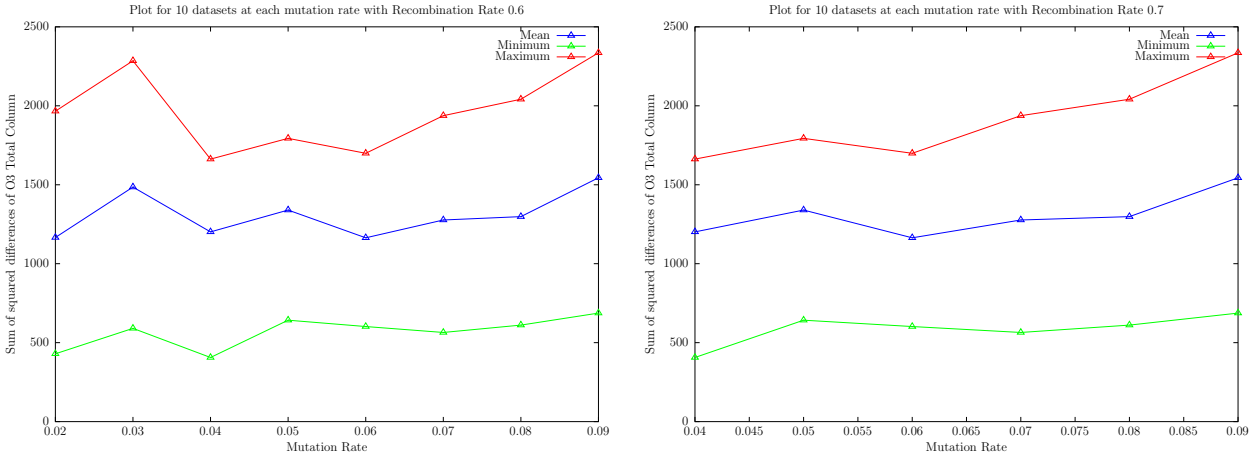
(a) Reference Input Vector (b) Random Input Vector

Figure 3.3 – Total Ozone Column

Figure 3.5(a) shows the result of the optimization process for using 40 out of 88 measurements, the residuals were minimized to a value of 429. Figure 3.5(b) shows the ratio between the reference and the estimate, which stayed stable over the whole number of 1428 measurements in the data set. Different values for the recombination and mutation rate were taken to determine how these values influence the results of the optimization. By applying these tests it was found out that the algorithm does not always converge to one global minimum. The resulting input vector varies often quite much, nevertheless the sum of the squared residuals stays for the example with 40 ones in a range between 400 and 2500.

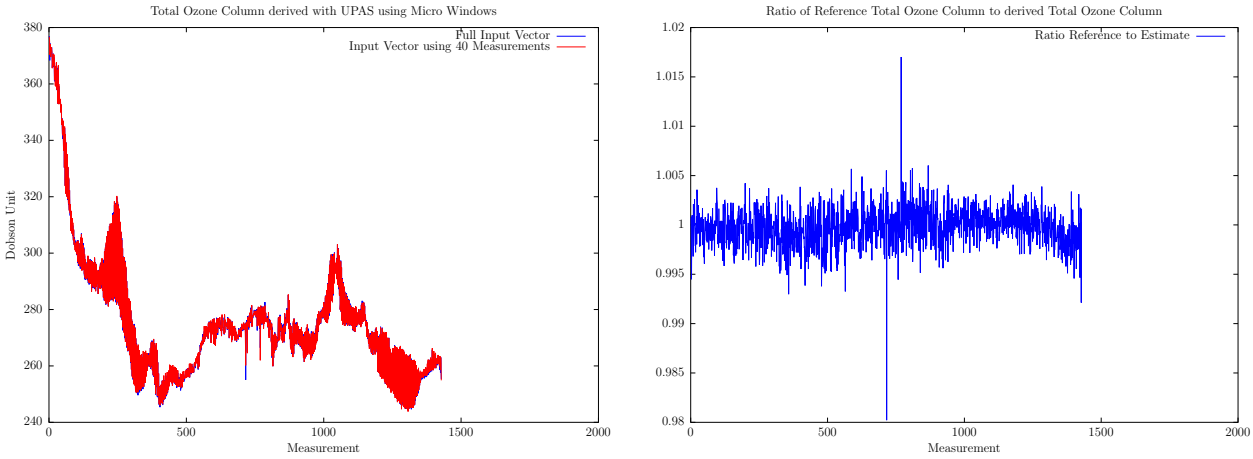
This means that

- the chosen values for recombination rate and mutation rate do not affect the result
- the algorithm don't converges to an global minimum



(a) Residuals for Recombination Rate 0.6 (b) Residuals for Recombination Rate 0.7

Figure 3.4 – Comparison of different Recombination and Mutation Rate combinations



(a) Residuals between Reference and Estimate (b) Ratio between Reference and Estimate

Figure 3.5 – Estimated Optimum

3.2 Optimization Results

The algorithm was applied with the restriction criterion of not using more then 20, 23, 29, 30, 35 and 40 measurements. According to the almost negligible influence of different mutation and recombination rates, as recombination rate the value 0.6 and as mutation rate 0.04 were chosen.

The algorithm was applied several times for each setup. Against the expectation the results were never equal. To separate at least between important and not that important measurements, the probability of being taken into account was computed with respect to the results of the algorithm. Figure 3.7(a) to 3.7(f) shows these distributions. The idea then was to take the most probable positions of the different input vectors to create the new vector which should be the global optimum. Unfortunately the assumption that by taking the most probable positions of ones in the input vector we can create an input vector with minimized residuals was wrong. It was seen that by doing so residuals were quite big, about the factor of 15 in comparison to the minimum residuals which were achieved by the algorithm. This leads also to the conviction that more than just the single positions the neighborhood relations between the single positions play a major role.

Another result is, that as expected by taking more measurements into account the residuals get smaller. Figure 3.6 gives a quite good picture of this. While with just 20 of 88 measurements the residuals range between 2200 and 8200, with more measurements both the minimum and maximum residual value decreases. Furthermore the range of the residuals shrinks with a raising number of measurements.

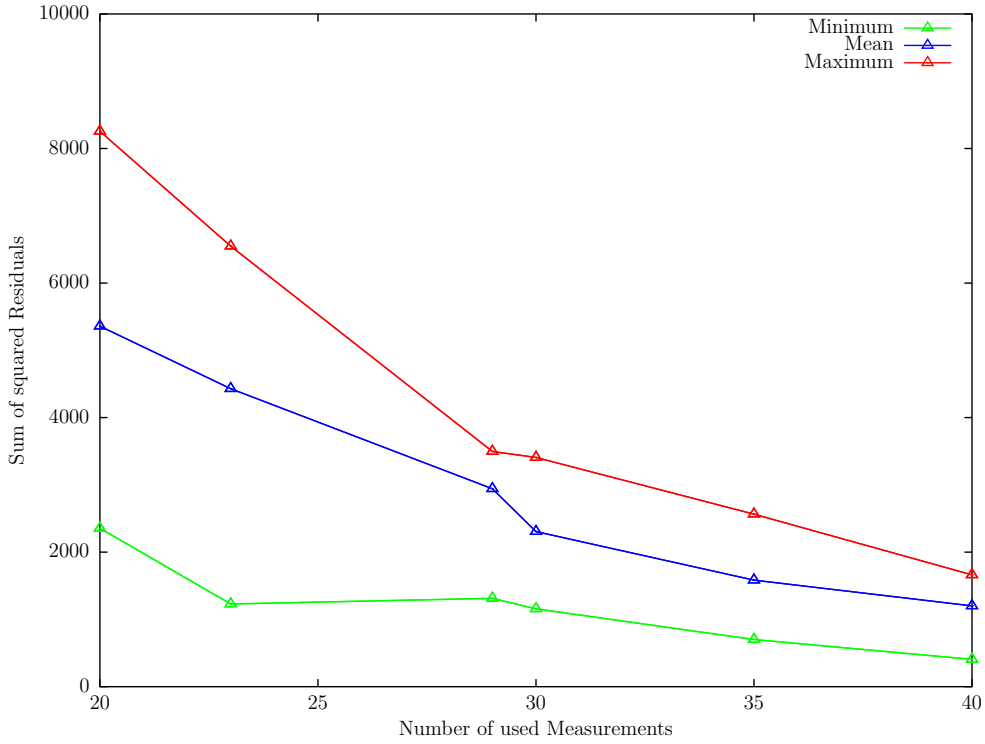
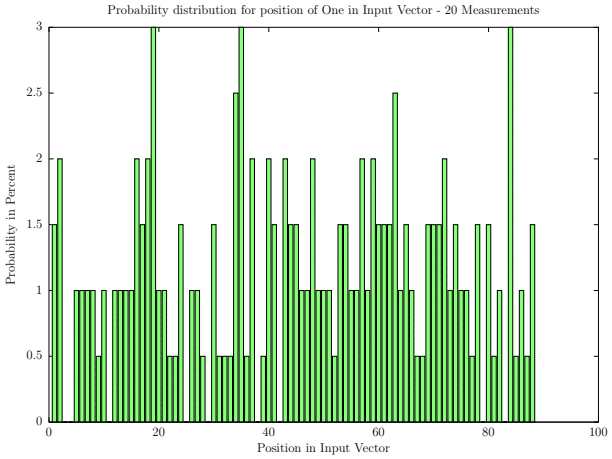
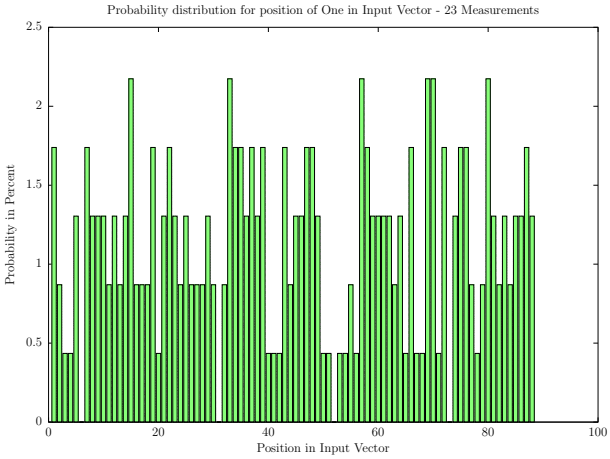


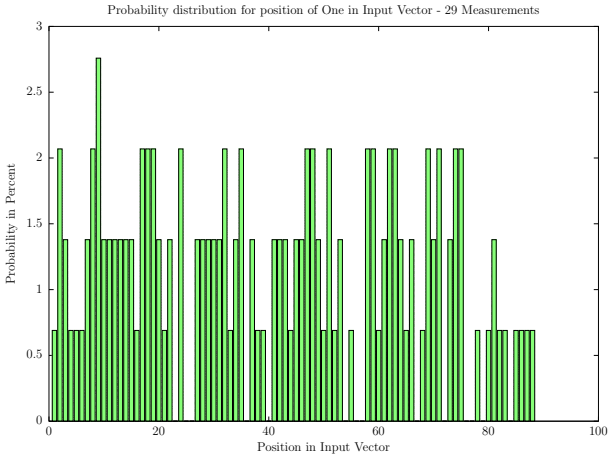
Figure 3.6 – Range of Residuals for different number of Measurements



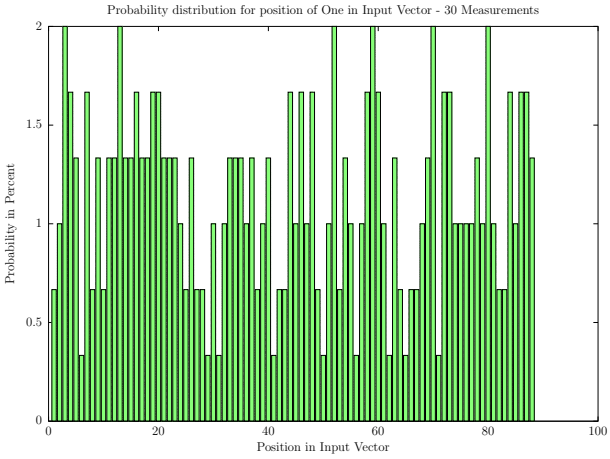
(a) Probability Analysis - 20 Measurements



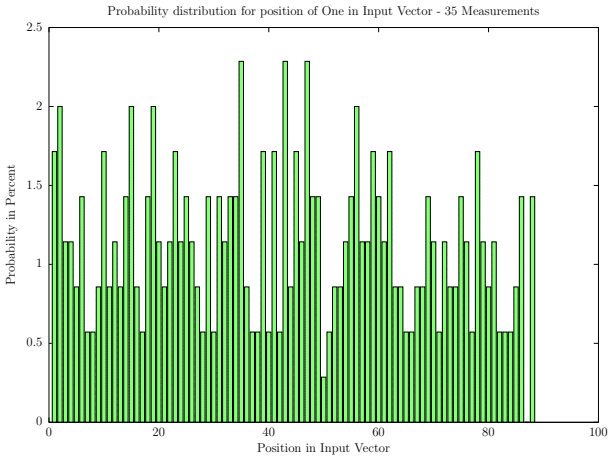
(b) Probability Analysis - 23 Measurements



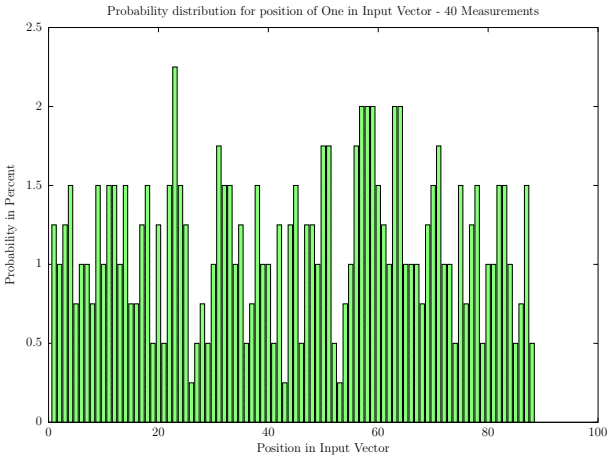
(c) Probability Analysis - 29 Measurements



(d) Probability Analysis - 30 Measurements



(e) Probability Analysis - 35 Measurements



(f) Probability Analysis - 40 Measurements

Figure 3.7 – Probability Analysis

4 Inversion of a cloud retrieval algorithm

The task of this chapter is the retrieval of two cloud parameters the cloud top height (CTH) and the cloud optical thickness (COT). A radiative transfer model (RTM) as forward model is employed. The RTM needs eight geophysical parameters to compute a spectra, as described in equation 2.2.4. As extension the RTM provides also the Jacobians for two parameters, CTH and COT. The problem is inverse stated. This means that the fitness function is formulated in the form $\min_{x\in\mathbb{R}^n} |y - f(\hat{x})|^2$. It was already shown that hybridization of GA leads to faster convergence and less function calls in contrast to pure GA. Because of the availability of partial derivatives, the genetic algorithm is enhanced with a local search algorithm which can make use of this additional information. A total least squares approach is used for local search according to Gauss-Newton-Method. The formula for iteratively updating CTH and COT is given with equation 4.1, the residuals are computed according to equation 4.2.

$$x_{i+1} = x_i - (A^T A)^{-1} A^T r \tag{4.1}$$

$$r = |y - f(x_i)|^2 \tag{4.2}$$

The measurements have similar accuracy, therefor no weighting has to be done. The local search algorithm was realized by using the CBLAS library as part of GSL. Both the genetic algorithm and the local search are forced to minimize the residuals between the estimated and the measured sprectra. Figure 4.1 shows in blue the measured spectra with the parameters CTH = 0.5 and COT = 0.113943. The red line shows a spectra derived with the genetic algorithm, the estimated values are CTH = 0.776452 and COT = 0.477708.

4.1 Algorithm design

The algorithm and its single components were fitted to the specific inversion problem as much as possible. Like in the previous chapter the main components are the three classes for population, chromosomes and genes. The population class includes now a local search method which can be applied to a single chromosome. The chromosomes two independent variables for CTH and COT are stored in a vector, the same is true for the dependent variable, the spectra. Because of the multi-threading, it was decided to declare the dependent and independent variables plus the fitness value as private and be only accessible via get and set operators. The genes contain

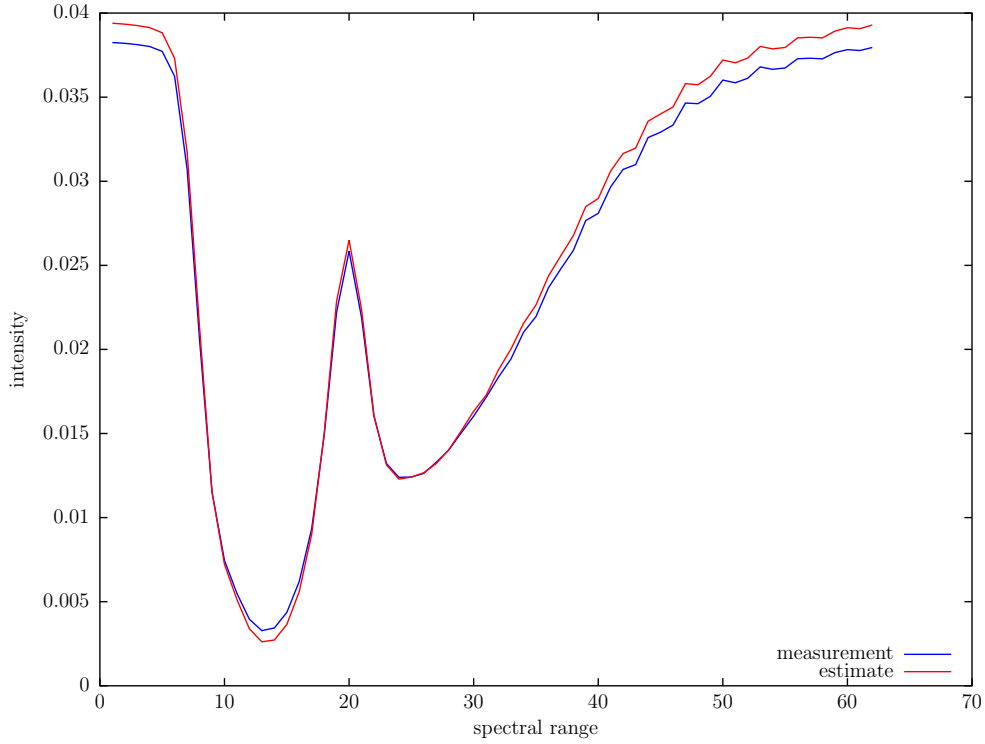


Figure 4.1 – Comparison between computed Spectra using estimated cloud parameters and measured spectra

methods for computing the independent value out the genetic string and for computing a genetic string out of an floating point number. The second is necessary because of the local search, which works with real numbers. The result of the local search than has to be converted into a genetic string. Figure 4.2 shows the setup of the classes.

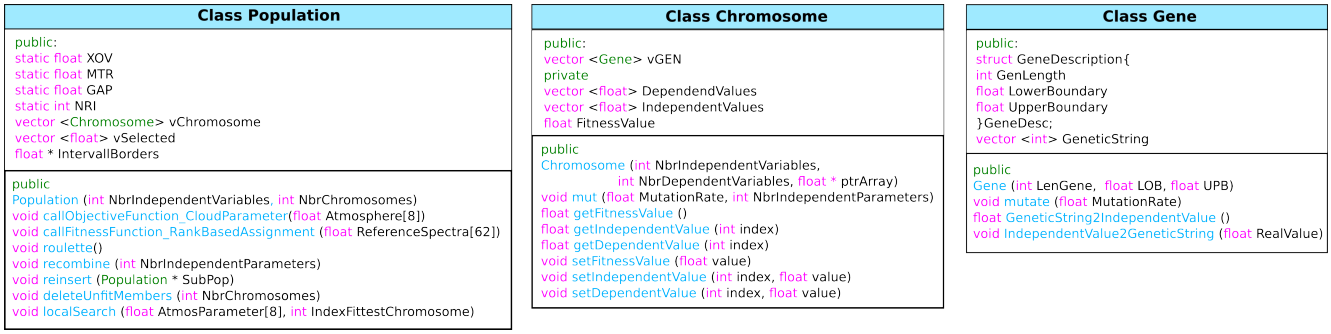


Figure 4.2 – Class diagram Clouds

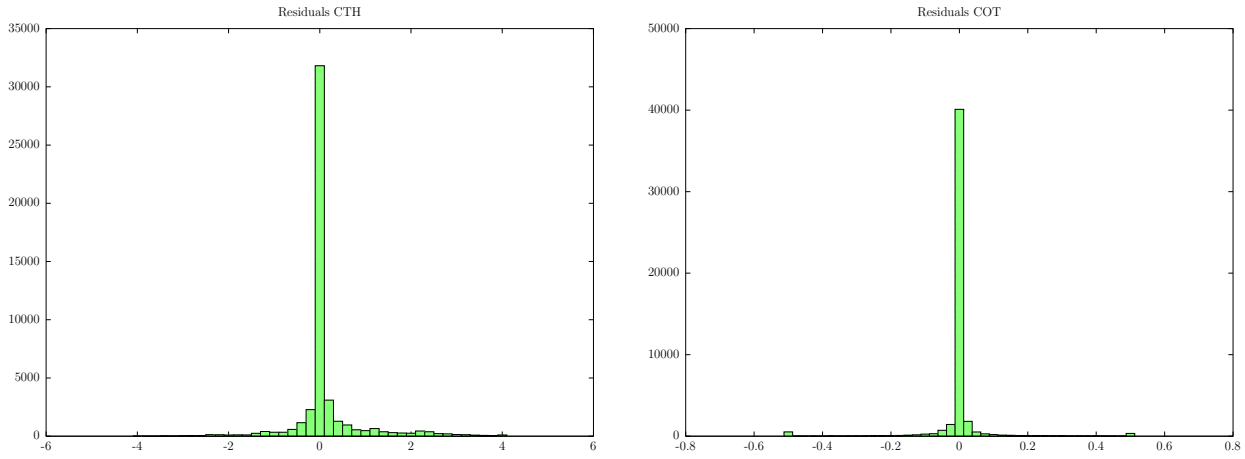
As stopping criterion stability in the independent variables for 14 loops was chosen. The local search was applied for maximum 10 iterations per generation, to ensure not to spend to much computational resources for the local search. Further steering parameters of the developed program are listed in table 4.1.

Parameter	**value**
Population size	50 chromosomes
Nbr. of generations	160
Generation gap	0.85
Mutation rate	0.07
Crossover rate	0.7
gene length	15
search space CTH	$x_i = [0.5 : 0.000274658 : 5]$
search space COT	$x_i = [0.11394 : 0.00012103 : 2.0969]$

Table 4.1 – parameters for testing with function inversion example

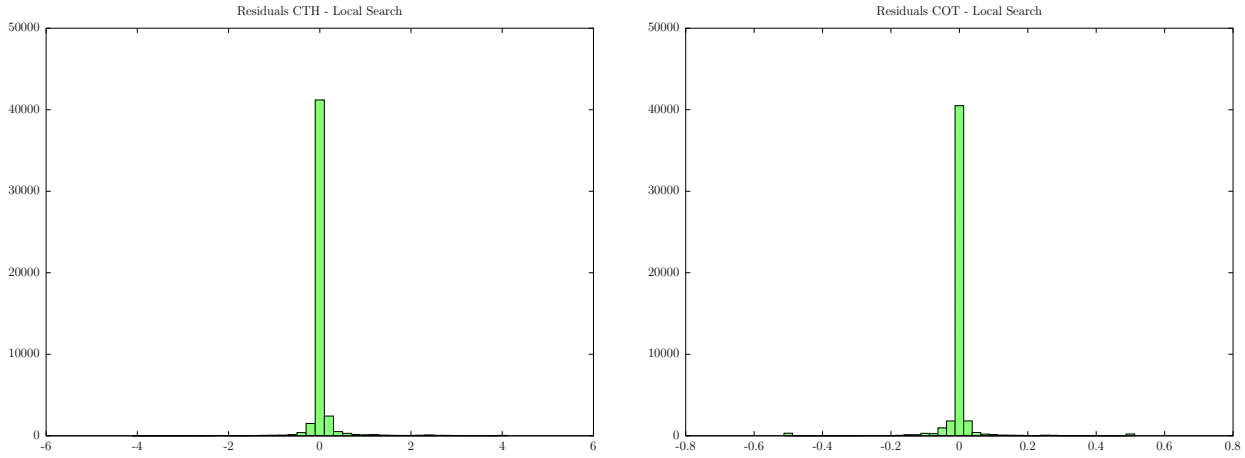
4.2 Optimization results

For assessing the quality of the retrieved parameters a representative data set of 48000 measurements is used. The algorithm was applied two times to this data set, one time without local search, the other time using local search. Figure 4.3 shows the residuals of the two retrieved parameters for the algorithm without local search, figure 4.3(a) for CTH and figure 4.3(b) for COT. Figure 4.4 shows the residuals of the same retrievals for the algorithm which included local search, figure 4.3(a) for CTH and figure 4.3(b) for COT. The root mean squared error (RMSE) was computed



(a) Residuals CTH (b) Residuals COT

Figure 4.3 – Residuals for CTH and COT retrieved with multi-threaded genetic algorithm



(a) Residuals CTH (b) Residuals COT

Figure 4.4 – Residuals for CTH and COT retrieved with multi-threaded hybrid genetic algorithm

for the two retrieval algorithms by

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y - f(\hat{x}))^2}{n}} \tag{4.3}$$

The RMSE of the genetic algorithm is for CTH = 0.74441 and for COT = 0.14837. The hybrid genetic algorithm performs better with RMSE for CTH = 0.37747 and for COT = 0.11852. This means the hybrid GA delivers for CTH an accuracy which is about the factor 1.97 higher than for the pure GA. Also for COT the hybrid GA achieves the higher accuracy, but here the factor is just 1.25.

In contrast to the previous results in section 2.3 the number of function calls used for the search is lower for the pure GA. It needs in mean about 3379 objective function calls until termination. The GA which makes use of local search needs in mean 3915.3 function calls until termination. This gives a ratio of 1 to 1.15 between pure GA and hybrid GA. The main reason for this is probably the stopping criterion for the local search algorithm, which terminates after 10 loops or if the difference between new and old CTH and COT is lower than 0.0000001. The local search needs 10 function calls, but the whole GA also just terminates if there is no change in CTH and COT of the fittest chromosome for 14 generations, so even this almost negligible changes in the local search force the hybrid GA to not terminate. In future the maximal accuracy determined by the genetic encoding of the variables should be taken into account here. By doing so, the number of function calls will probably decrease for the hybrid GA without loosing accuracy.

5 Conclusion

Several conclusions can be made out of this project. From the methodological side the prove is made that EA fit well for a wide range of optimization and search problems. Different forward and inverse problems were discussed in detail. The quality of solution depends at least mostly on how well the problem has been discussed and analyzed. A priori knowledge on the objective function behavior is the key to produce reasonable results. Therefor the user has to know how the different operators of EA influence the process. With this knowledge the user can set the probabilities dependent on the problem to guide the search.

But besides of the theoretical side, the main focus lies on the results the program delivers for the real world problems. The first task was to find an optimal input vector for the ozone algorithm with the restriction of not using more than a given number of measurements. This task was from computational side quite complicated, the problem is in practice not solvable by a deterministic approach because of the needed runtime. Here heuristic approaches like the developed GA convince. Combinatorical problems fit almost ideal for genetic algorithms, because of the feature to deliver results which are almost optimal in short time. By adding parallelization to the algorithm, the run time was reduced even more. The program was running on a blade server with 48 cores exclusive for the program. This meant a run time reduction by the factor of 48 in contrast to a sequential program. The algorithm did not always converge to the same results. This behavior is explained by the nature of combinatorical problems. An probability driven approach was done in order to retrieve even better fitting input vectors out of the single results of the program. It was shown that this approach don't works. This leads to the conviction that despite of the single positions of the used measurements furthermore neighborhood relationship between the single measurements play a role. Further research can be done here in investigating these relationships.

For the inversion of a complex function the algorithm works successful. High quality results were achieved on a reliable base. Despite of the parallelization for run time reduction, a local search algorithm was added. Because of the inverse formulation of the problem, in contrast to minimization algorithms for forward models a total least squares approach was introduced. It is shown that the retrieved results of this hybrid algorithm are for CTH about the factor 1.97 that precise than the one of the pure genetic algorithm, for COT the accuracy of the estimation is about the factor 1.25 more precise.

Bibliography

[1] Hestenes, M.R. and Stiefel, E. (1952): Methods of conjugate gradients for solving linear systems. Journal of Research of the National Bureau of Standards, 49, pp. 409-439.

[2] Fletcher, R. and Reeves, C.M. (1964): Function minimization by conjugate gradients. Computer Journal, 7, pp. 149-154.

[3] Nocedal, J. and Wright, S.J. (2006): Numerical Optimization - Second Edition. Springer. New York, USA.

[4] Schröder, Dierk. (2010): Intelligente Verfahren: Identifikation und Regelung nichtlinearer Systeme. Springer Verlag. Heidelberg, Deutschland.

[5] Papula, Lothar. (2001): Mathematik für Ingenieure und Naturwissenschaftler - Band 1, 10. Auflage. Vieweg Verlag. Braunschweig/Wiesbaden, Deutschland.

[6] Weicker, Karsten. (2002): Evolutionäre Algorithmen. Teubner Verlag. Deutschland.

[7] Fogel, D.B. (1995): Evolutionary Computation: Toward a New Philosophy of Machine Intelligence . Piscataway, NJ: IEEE Press.

[8] Chipperfield, A., Fleming, P. J., Pohlheim, H. and Fonseca, C. M. (1994): Genetic Algorithm Toolbox for use with Matlab. Technical Report No. 512, Department of Automatic Control and Systems Engineering, University of Sheffield.

[9] De Jong, K.A. (1975): Analysis of the Behaviour of a Class of Genetic Adaptive Systems. PhD Thesis, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor.

[10] Baker, J.E. (1987): Reducing bias and inefficiency in the selection algorithm. Proc. ICGA 2, pp. 14-21

[11] Goldberg, D.E. (1989): Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley Publishing Company.

[12] Whitley, D. (1989): The GENITOR algorithm and selection pressure: why rank-based allocations of reproductive trials is best. Proc. ICGA 3, pp. 116 - 121.

[13] Huang, R. and Fogarty, T. C. (1991): Adaptive Classification and Control-Rule Optimization Via a Learning Algorithm for Controlling a Dynamic System. Proc. 30th Conf. Decision and Control, Brighton, England, pp. 867 - 868.

[14] Bagheri, E. and Deldari, H. (2006): Dejong Function Optimization by means of a Parallel Approach to Fuzzified Genetic Algorithm. Proc. 11. IEEE Symposium on Computers and Communications, Cagliari, Sardinia, Italy, pp. 675 - 680.

[15] Reed, R.D. and Marks, R.J. (1995): An Evolutionary Algorithm for Function Inversion and Boundary Marking. Volume 2, IEEE International Conference on Evolutionary Computation, pp. 794 - 797.

[16] Heuser, H. (1990): Lehrbuch der Analysis, Teil 1. Volume 8, Teubner Verlag, Stuttgart.

[17] Shekel, J. (1971): Test functions for multimodal search techniques. Fifth Annual Princeton Conference on Information Science and Systems.

[18] Gordon V.S. and Whitley D (1993): Serial and parallel genetic algorithms as function optimizers. In: Forrest S. (Ed.), Proceedings of the Fifth International Conference of Genetic Algorithms, Morgen Kaufmann, San Mateo, CA, pp. 177 - 183.

[19] Baluja S. (1993): Structure and performance of fine-grain parallelism in genetic search. In: Forrest S. (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp. 114 -121.

[20] Hart W.E., Baden S., Belew R.K., and Kohn S. (1997) Analysis of the numerical effects of parallelism on a parallel genetic algorithm. Proceedings of the Worksphop on Solving Combinatorial Optimization Problems in Parallel. IEEE (Ed.), CD-ROM IPPS97.

[21] Tsoulos I. and Lagaris I.E. (2008): GenMin: An enhanced genetic algorithm for global optimization. Computer Physics Communications, doi: 10:1016/j.cpc.2008.01.040.

[22] Bashir H.A. and Neville R.S. (2012): A Hybrid Evolutionary Computation Algorithm for Global Optimization. IEEE World Congress on Computational Intelligence, June, 10-15, 1012 - Brisbane, Australia.

[23] Alba E. and Troya J.M. (2002): Improving flexibility and efficiency by adding parallelism to genetic algorithms. Statistics and Computing 12: 91-114, Kluwer Academic Publishers, Netherlands.

[24] Zhang J., Zhan Z., Lin Y., Chen N., Gong Y., Zhong J., Chung H., Li Y., Shi Y. (2011): Evolutionary Computations Meets Machine Learning: A Survey. IEEE Computational Intelligence Magazine, doi: 10.1109/MCI.2011.942584.

[25] Internetrecherche (2012) http://en.wikipedia.org/wiki/Convex_function date: 17.09.2012

[26] Internetrecherche (2012) http://en.wikipedia.org/wiki/Unimodal_function#Unimoda date: 17.09.2012

[27] Internetrecherche (2012) http://en.wikipedia.org/wiki/Quadratic_function date: 17.09.2012